**Государственное бюджетное профессиональное образовательное учреждение «Нижегородский Губернский колледж»**

Методическая комиссия «Информатика и вычислительная техника»

Допущен к защите:

преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_ Л.В. Мухина,

\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Голубева,

«21»\_\_марта\_\_2025 г.

**ОТЧЕТ** **ПО**

**УЧЕБНОЙ ПРАКТИКЕ**

ПМ.01 РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

Руководитель: \_\_\_\_\_\_\_\_\_\_ Мухина Л.В., Голубева Е.П. 21.03.2025 г.

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Климычева Анастасия Артёмовна 21.03.2025 г.

Специальность, группа: 09.02.07, 43П

Нижний Новгород

2025 г.

**Содержание**

[Введение 3](#_Toc193420129)

[Разработка информационной системы Мастер пол 5](#_Toc193420130)

[Разработка базы данных 5](#_Toc193420131)

[Архитектура проекта и используемые технологии 6](#_Toc193420132)

[Вывод списка партнёров 8](#_Toc193420133)

[Экран редактирования партнёра 10](#_Toc193420134)

[Экран просмотра истории продаж партнёра 13](#_Toc193420135)

[Экран расчёта количества материала 15](#_Toc193420136)

[Разработка Dll 18](#_Toc193420137)

[Тестирование 21](#_Toc193420138)

[Разработка UML диаграмм. 25](#_Toc193420139)

[Разработка информационной системы «Пиши-Стирай» 27](#_Toc193420140)

[Разработка базы данных, объектов базы данных и импорт 27](#_Toc193420141)

[Разработка настольного приложения 32](#_Toc193420142)

[Разработка библиотеки 37](#_Toc193420143)

[Тестирование библиотеки 38](#_Toc193420144)

[Тестовая документация 40](#_Toc193420145)

[Разработка информационной системы Matule Me 46](#_Toc193420146)

[Разработка серверной части 46](#_Toc193420147)

[Используемые библиотеки 47](#_Toc193420148)

[Архитектура приложения 49](#_Toc193420149)

[Модуль авторизации 51](#_Toc193420150)

[Основной модуль 57](#_Toc193420151)

[Заключение 64](#_Toc193420152)

[Приложения 65](#_Toc193420153)

[Приложение А. 65](#_Toc193420154)

[Приложение Б. 67](#_Toc193420155)

[Приложение В. 69](#_Toc193420156)

[Приложение Г. 71](#_Toc193420157)

[Приложение Д. 75](#_Toc193420158)

# **Введение**

Учебная практика по ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» проходила в ГБПОУ «Нижегородский Губернский колледж» в период с 01.03.2025 по 21.03.2025 г.

В современном мире информационные технологии играют ключевую роль во всех сферах жизни: от бизнеса и промышленности до образования и медицины. Разработка программного обеспечения (ПО) для компьютерных систем является неотъемлемой частью цифровизации и автоматизации процессов. Учебная практика по данной дисциплине актуальна по следующим причинам:

1. Высокий спрос на разработчиков ПО. Современные компании нуждаются в квалифицированных специалистах, способных разрабатывать и поддерживать программные решения для различных компьютерных систем. Обучение практическим навыкам разработки модулей ПО позволяет студентам подготовиться к реальной работе в IT-сфере.

2. Развитие модульного подхода в программировании. В настоящее время разработка ПО строится на принципах модульности, что упрощает создание, тестирование и поддержку программных продуктов. Владение методами модульного программирования является важным навыком для будущих специалистов.

3. Актуальность использования современных технологий. Практика охватывает изучение языков программирования, инструментов разработки, методов тестирования и отладки программного кода. Это позволяет студентам освоить востребованные технологии, используемые в реальных проектах.

Таким образом, учебная практика «Разработка модулей программного обеспечения для компьютерных систем» является актуальной и востребованной, так как способствует подготовке квалифицированных специалистов, способных разрабатывать современные программные продукты и решать актуальные задачи IT-индустрии.

Целью данной учебной практики является овладение практическими навыками разработки, тестирования и отладки модулей программного обеспечения, а также освоение современных методов и инструментов программирования для создания эффективных и надёжных программных решений, включая мобильные приложения.

Для достижения данной цели поставлены следующие задачи:

1) принципы модульного программирования и ознакомиться с архитектурой программных систем и принципами их построения;

2) Ознакомиться с принципами объектно-ориентированного программирования.

3) Разработать программные модули на языке программирования C# на кроссплатформенном XAML фреймворке для платформы .NET – Avalonia;

4) Овладеть основами мобильной разработки, изучить инструменты и среды разработки для создания мобильных приложений;

5) Использовать современные инструменты и среды разработки. Реализовать взаимодействия между модулями ПО;

6) Применить модульное и интеграционное тестирование. Использовать системы контроля версий (Git). Практическим результатом прохождения учебной практики является отчёт по выполнению заданий с представленными в нём скриншотами, программными кодами и диаграммами.

# **Разработка информационной системы Мастер пол**

## **Разработка базы данных**

На основе описания предметной области создана база данных в СУБД Postgres для разрабатываемой системы, содержащая только нужные таблица и связи для реализации системы. База данных находится в 3 нормальной форме с обеспечением ссылочной целостности.

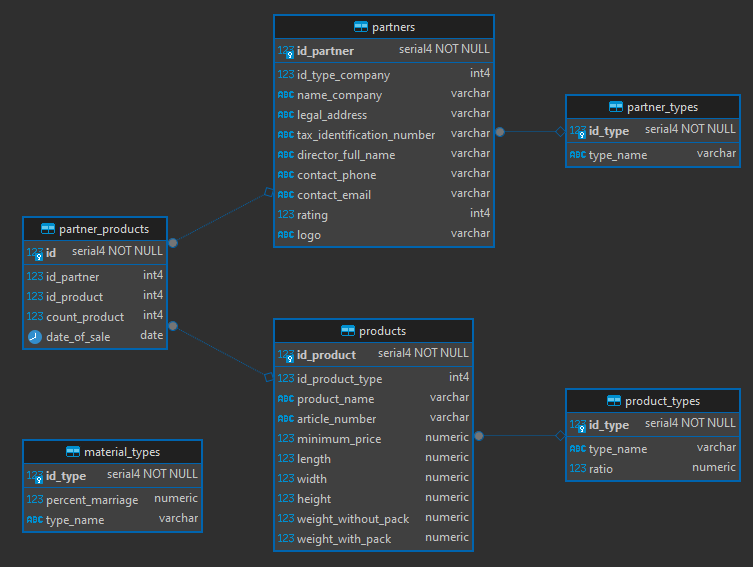


Рисунок №1 – Схема базы данных

Скрипт создания БД представлен в Приложении А, таблице 1.

## **Архитектура проекта и используемые технологии**

В разработанном приложении используется архитектура MVVM (Model-View-ViewModel), которая является одной из самых популярных архитектур для создания приложений с пользовательским интерфейсом. MVVM разделяет логику приложения на три основных компонента:

1. Model (Модель): этот компонент отвечает за бизнес-логику и управление данными. В нашем приложении модель реализована с использованием Entity Framework, который обеспечивает удобный доступ к базе данных и управление данными через объектно-ориентированный подход. Entity Framework позволяет работать с данными как с объектами, что упрощает взаимодействие с базой данных и уменьшает количество кода, необходимого для выполнения операций CRUD (создание, чтение, обновление, удаление).
2. View (Представление): представление отвечает за отображение данных пользователю и взаимодействие с ним. В нашем случае мы используем Avalonia, кроссплатформенный фреймворк для создания пользовательских интерфейсов. Avalonia позволяет создавать современные и отзывчивые интерфейсы, поддерживающие различные платформы, такие как Windows, macOS и Linux. Это обеспечивает широкую доступность приложения для пользователей.
3. ViewModel (Модель представления): ViewModel служит связующим звеном между моделью и представлением. Он обрабатывает пользовательский ввод, взаимодействует с моделью и обновляет представление. ViewModel реализует интерфейсы INotifyPropertyChanged и ICommand, что позволяет автоматически обновлять интерфейс при изменении данных и обрабатывать команды от пользователя.

Используемые технологии:

1. Avalonia: это кроссплатформенный фреймворк для создания пользовательских интерфейсов на .NET. Он поддерживает XAML для разметки интерфейса и предоставляет мощные инструменты для создания отзывчивых и современных приложений.
2. Entity Framework: это ORM (Object-Relational Mapping) библиотека для .NET, которая упрощает работу с базами данных. Она позволяет разработчикам работать с данными в виде объектов, а не в виде таблиц и строк, что значительно упрощает код и повышает его читаемость.
3. MessageBoxAvalonia: эта библиотека предоставляет функциональность для отображения диалоговых окон сообщений в приложениях на Avalonia. Она позволяет легко создавать и настраивать диалоговые окна для отображения информации пользователю или запроса подтверждения.
4. .NET 8: это последняя версия платформы .NET, которая включает в себя множество улучшений производительности, новых возможностей и обновлений для разработки приложений. Использование .NET 8 позволяет нам использовать современные функции языка C# и улучшенные библиотеки.
5. Community Toolkit: это набор инструментов и библиотек, разработанных для упрощения разработки приложений на .NET. Он включает в себя различные вспомогательные классы и функции, которые помогают в реализации паттернов проектирования и упрощают работу с данными и пользовательским интерфейсом.

В результате использования архитектуры MVVM и современных технологий, таких как Avalonia и Entity Framework, приложение обладает высокой модульностью, простотой в сопровождении и расширяемостью. Это позволяет легко вносить изменения и добавлять новые функции, а также обеспечивает удобный и интуитивно понятный интерфейс для пользователей.

## **Вывод списка партнёров**

В данном разделе рассмотрим настройку экрана вывода списков продуктов. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает четкое разделение логики приложения и пользовательского интерфейса.

1. Структура ViewModel:

* Класс PartnersListVM: это основной класс ViewModel, который отвечает за управление данными и логикой, связанной со списком партнеров (в данном случае, продуктов).
* Свойство \_partners: это приватное поле, помеченное атрибутом [ObservableProperty], которое хранит список объектов типа Partner. Это свойство автоматически уведомляет представление об изменениях, что позволяет обновлять интерфейс при изменении данных.
* Конструктор PartnersListVM: в конструкторе происходит инициализация списка партнеров. Данные загружаются из контекста базы данных с использованием Entity Framework, включая связанные данные (например, тип компании и продукты партнера) с помощью метода Include.

|  |
| --- |
| public PartnersListVM()  {  Partners = MainWindowViewModel.Context.Partners  .Include(it => it.IdTypeCompanyNavigation)  .Include(it => it.PartnerProducts)  .ToList();  } |

1. Команды для взаимодействия с пользователем

Создание, редактирование и удаление партнеров: ViewModel содержит несколько команд, которые позволяют пользователю взаимодействовать с данными:

* CreatePartner: открывает экран для создания нового партнера.
* EditPartner: открывает экран редактирования выбранного партнера.
* DeletePartner: удаляет выбранного партнера после подтверждения от пользователя через диалоговое окно.

|  |
| --- |
| [RelayCommand]  public async void DeletePartner(Partner partner)  {  // Логика удаления партнера с подтверждением  } |

1. Обработка событий и взаимодействие с интерфейсом

Использование MessageBox: для подтверждения действий пользователя (например, удаление партнера) используется библиотека MessageBoxAvalonia, которая позволяет отображать диалоговые окна с сообщениями.

Обновление текущей страницы: после выполнения команд, таких как создание или удаление партнера, ViewModel обновляет текущую страницу в MainWindowViewModel, что позволяет пользователю видеть актуальный список партнеров.

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойству Partners из ViewModel. Это позволяет автоматически отображать список партнеров в интерфейсе, используя элемент управления ListBox.

|  |
| --- |
| <ListBox ItemsSource="{Binding Partners}">  <ListBox.ItemTemplate>  <DataTemplate>  <!-- Определение шаблона для отображения данных -->  </DataTemplate>  </ListBox.ItemTemplate>  </ListBox> |

## **Экран редактирования партнёра**

В данном разделе рассмотрим настройку экрана редактирования партнера. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает удобное взаимодействие пользователя с данными.

1. Структура ViewModel:

Класс PartnerEditorVM: это ViewModel, который отвечает за управление данными и логикой, связанной с созданием и редактированием партнера.

Свойства:

* NameScreen: строка, определяющая заголовок экрана. В зависимости от контекста (создание или редактирование) она может принимать значения "Добавление партнёра" или "Редактирование партнёра".
* NameButton: строка, определяющая текст на кнопке. Она также меняется в зависимости от контекста.
* \_partner: приватное поле, помеченное атрибутом [ObservableProperty], которое хранит объект типа Partner. Это свойство автоматически уведомляет представление об изменениях.
* \_partnerTypes: список типов партнеров, доступных для выбора.
* \_selPartnerType: выбранный тип партнера, который также помечен как наблюдаемое свойство.

|  |
| --- |
| [ObservableProperty]  private Partner \_partner = new Partner(); |

1. Конструкторы ViewModel

Конструктор без параметров: инициализирует данные для нового партнера, устанавливая заголовок экрана и текст кнопки на "Добавление партнёра" и "Добавить" соответственно. Также загружает доступные типы партнеров из базы данных.

|  |
| --- |
| public PartnerEditorVM()  {  InitData();  SelPartnerType = PartnerTypes.FirstOrDefault();  } |

Конструктор с параметром idPartner: используется для редактирования существующего партнера. Заголовок экрана и текст кнопки изменяются на "Редактирование партнёра" и "Редактировать". Данные партнера загружаются из базы данных по его идентификатору.

|  |
| --- |
| public PartnerEditorVM(int idPartner)  {  NameScreen = "Редактирование партнёра";  NameButton = "Редактировать";  InitData();  Partner = MainWindowViewModel.Context.Partners.First(it => it.IdPartner == idPartner);  SelPartnerType = Partner.IdTypeCompanyNavigation;  } |

1. Инициализация данных

Метод InitData: загружает список типов партнеров из базы данных и инициализирует свойство PartnerTypes. Это позволяет пользователю выбирать тип партнера из доступных вариантов.

|  |
| --- |
| private void InitData()  {  PartnerTypes = MainWindowViewModel.Context.PartnerTypes.ToList();  } |

1. Команды для взаимодействия с пользователем

Save: эта команда обрабатывает сохранение данных партнера. Если идентификатор партнера равен 0, это означает, что создается новый партнер, и он добавляется в базу данных. В противном случае, данные существующего партнера обновляются. После успешного сохранения пользователь получает уведомление через диалоговое окно.

|  |
| --- |
| [RelayCommand]  public async void Save()  {  try  {  Partner.IdTypeCompanyNavigation = SelPartnerType;  if (Partner.IdPartner == 0)  {  MainWindowViewModel.Context.Partners.Add(Partner);  MainWindowViewModel.Context.SaveChanges();  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  await MessageBoxManager.GetMessageBoxStandard("Сообщение", $"Партнёр '{Partner.NameCompany}' был успешно добавлен", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  else  {  MainWindowViewModel.Context.SaveChanges();  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  await MessageBoxManager.GetMessageBoxStandard("Сообщение", $"Партнёр '{Partner.NameCompany}' был изменён", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  }  catch (Exception ex)  {  await MessageBoxManager.GetMessageBoxStandard("Ошибка", $"{ex.Message}", MsBox.Avalonia.Enums.ButtonEnum.Ok).ShowAsync();  }  } |

GoBack: Эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без сохранения изменений.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойствам ViewModel. Это позволяет автоматически отображать данные партнера и типы партнеров в интерфейсе, используя элементы управления, такие как TextBox для ввода данных и ComboBox для выбора типа партнера.

|  |
| --- |
| <TextBox Text="{Binding Partner.NameCompany}" />  <ComboBox ItemsSource="{Binding PartnerTypes}" SelectedItem="{Binding SelPartnerType}" /> |

## **Экран просмотра истории продаж партнёра**

В данном разделе рассмотрим настройку экрана истории партнера.

1. Структура ViewModel

Класс HistoryPartnerVM: это ViewModel, который отвечает за управление данными и логикой, связанной с отображением истории реализации продукции для конкретного партнера.

* Свойство NameScreen: Строка, определяющая заголовок экрана. По умолчанию устанавливается значение "История".
* \_listPartProds: Приватное поле, помеченное атрибутом [ObservableProperty], которое хранит список объектов типа PartnerProduct. Это свойство автоматически уведомляет представление об изменениях, что позволяет обновлять интерфейс при изменении данных.

|  |
| --- |
| [ObservableProperty]  private List<PartnerProduct> \_listPartProds = new(); |

1. Конструктор ViewModel

Конструктор с параметром id: этот конструктор используется для инициализации ViewModel с идентификатором партнера. Он загружает историю реализации продукции для данного партнера из базы данных и формирует заголовок экрана, включая название компании партнера.

|  |
| --- |
| public HistoryPartnerVM(int id)  {  ListPartProds = MainWindowViewModel.Context.PartnerProducts  .Where(it => it.IdPartner == id)  .Include(it => it.IdProductNavigation)  .ToList();  NameScreen = $"История реализации продукции партнёра \"{MainWindowViewModel.Context.Partners.FirstOrDefault(it => it.IdPartner == id).NameCompany}\"";  } |

1. Загрузка данных

Загрузка истории продукции: в конструкторе происходит выборка данных о продукции, связанной с конкретным партнером, из базы данных. Используется метод Include для загрузки связанных данных о продуктах, что позволяет получить полную информацию о каждой записи.

1. Команды для взаимодействия с пользователем

GoBack: эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без каких-либо изменений. Она обновляет текущую страницу в MainWindowViewModel.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойству ListPartProds из ViewModel. Это позволяет автоматически отображать список истории реализации продукции в интерфейсе, используя элемент управления ListBox.

|  |
| --- |
| <ListBox ItemsSource="{Binding ListPartProds}">  <!-- Определение шаблона для отображения данных о продукции -->  </ ListBox > |

## **Экран расчёта количества материала**

В данном разделе рассмотрим настройку экрана расчета материалов. Экран реализован с использованием архитектуры MVVM и фреймворка Avalonia, что обеспечивает удобное взаимодействие пользователя с данными для расчета необходимых материалов.

1. Структура ViewModel

Класс CalculationMaterialVM: это ViewModel, который отвечает за управление данными и логикой, связанной с расчетом количества материалов, необходимых для производства продукции.

Свойства:

* \_quantityProduct: Приватное поле, помеченное атрибутом [ObservableProperty], которое хранит количество продукции, для которой будет производиться расчет.
* \_materialTypes: Список типов материалов, доступных для выбора.
* \_products: Список продуктов, для которых будет производиться расчет.
* \_materialType: Выбранный тип материала, который также помечен как наблюдаемое свойство.
* \_product: Выбранный продукт, для которого будет производиться расчет.
* \_result: Результат расчета, который будет отображаться пользователю.

|  |
| --- |
| [ObservableProperty]  private int \_quantityProduct = 0; |

1. Конструктор ViewModel

Конструктор CalculationMaterialVM: в этом конструкторе происходит инициализация данных для расчета. Загружаются доступные типы материалов и продукты из базы данных с использованием Entity Framework. Также устанавливаются значения по умолчанию для выбранного типа материала и продукта.

|  |
| --- |
| public CalculationMaterialVM()  {  MaterialTypes = MainWindowViewModel.Context.MaterialTypes.ToList();  Products = MainWindowViewModel.Context.Products.Include(it => it.IdProductTypeNavigation).ToList();  MaterialType = MaterialTypes.FirstOrDefault();  Product = Products.FirstOrDefault();  } |

1. Команды для взаимодействия с пользователем

Calculate: эта команда выполняет расчет необходимого количества материалов на основе введенных данных. Она использует класс CalculationClass, который содержит логику для вычисления количества материалов, необходимых для производства заданного количества продукции. Результат расчета сохраняется в свойстве Result.

|  |
| --- |
| [RelayCommand]  public void Calculate()  {  Result = \_calculation.GetQuantityForProduct((int)Product.IdProductType, MaterialType.IdType, QuantityProduct, (double)Product.Height, (double)Product.Width);  } |

GoBack: эта команда позволяет пользователю вернуться на предыдущий экран (список партнеров) без каких-либо изменений. Она обновляет текущую страницу в MainWindowViewModel.

|  |
| --- |
| [RelayCommand]  public void GoBack()  {  MainWindowViewModel.Instance.CurrentPage = new PartnersListView();  } |

1. Инициализация расчетного класса

Класс CalculationClass: Этот класс содержит логику для расчета количества материалов на основе различных параметров, таких как тип продукта, тип материала, количество продукции и размеры продукта. Он использует списки для хранения коэффициентов и процентов брака, что позволяет гибко управлять расчетами.

1. Привязка данных к представлению

XAML (представление): в XAML-файле используется привязка данных к свойствам ViewModel. Это позволяет автоматически отображать данные о типах материалов, продуктах и результатах расчетов в интерфейсе, используя элементы управления, такие как ComboBox для выбора типа материала и продукта, а также TextBox для ввода количества продукции.

|  |
| --- |
| <ComboBox ItemsSource="{Binding MaterialTypes}" SelectedItem="{Binding MaterialType}" />  <ComboBox ItemsSource="{Binding Products}" SelectedItem="{Binding Product}" />  <TextBox Text="{Binding QuantityProduct}" />  <TextBlock Text="{Binding Result}" /> |

## **Разработка Dll**

В данном разделе описывается процесс разработки динамической библиотеки (DLL), которая включает класс CalculationClass. Этот класс предназначен для выполнения расчетов необходимого количества материалов для производства продукции с учетом возможного брака. Библиотека используется в приложении для обеспечения точности расчетов.

1. Цели и задачи разработки DLL

Целью разработки данной DLL является создание модуля, который предоставляет функционал для расчета необходимого количества материалов на основе различных параметров. Основные задачи включают:

* Реализация класса CalculationClass, который содержит метод для расчета количества материалов.
* Обеспечение корректной работы метода с учетом возможных ошибок и исключительных ситуаций.
* Подготовка библиотеки к интеграции с основным приложением.

1. Структура DLL

DLL состоит из одного основного класса — CalculationClass, который включает:

* Списки типов материалов и типов продукции.
* Метод GetQuantityForProduct, который выполняет расчеты.

1. Реализация класса CalculationClass

Класс CalculationClass реализован следующим образом:

|  |
| --- |
| public class CalculationClass  {  // Список типов материалов с процентом брака  private List<(int IdType, double PercentMarriage)> \_materialTypes = new()  {  (1, 0.1), // 10% брака  (2, 0.95), // 95% брака  (3, 0.28), // 28% брака  (4, 0.55), // 55% брака  (5, 0.34) // 34% брака  };  // Список типов продукции с коэффициентами  private List<(int IdType, double Ratio)> \_productTypes = new()  {  (1, 2.35),  (2, 5.15),  (3, 4.34),  (4, 1.5)  };  // Конструктор класса  public CalculationClass() { }  // Метод для расчета необходимого количества материала  public int GetQuantityForProduct(int typeProduct, int typeMaterial, int quantityProduct, double weight, double height)  {  int quantityMaterial = 0;  double percent\_marriage = 0, ratio = 0;  // Проверка на существование типов и корректность входных данных  if (!\_materialTypes.Any(it => it.IdType == typeMaterial) ||  !\_productTypes.Any(it => it.IdType == typeProduct) ||  quantityProduct < 0 || weight < 0 || height < 0)  {  return -1; // Возврат -1 в случае ошибок  }  // Получение процента брака и коэффициента типа продукции  percent\_marriage = \_materialTypes.First(it => it.IdType == typeMaterial).PercentMarriage;  ratio = \_productTypes.First(it => it.IdType == typeProduct).Ratio;  // Расчет количества материала для одной единицы продукции  int quantityFor1Product = (int)((weight \* height) \* ratio);  // Общий расчет необходимого материала с учетом брака  quantityMaterial = (int)(quantityFor1Product \* quantityProduct \* (1.0 + percent\_marriage));  return quantityMaterial; // Возврат рассчитанного количества материала  }  } |

1. Логика метода GetQuantityForProduct

Проверка входных данных: метод проверяет, существуют ли указанные типы продукции и материалов, а также корректны ли значения для количества продукции и параметров (вес и высота). Если данные некорректны, метод возвращает -1.

Получение данных: если входные данные корректны, метод извлекает процент брака и коэффициент типа продукции из соответствующих списков.

Расчет: метод вычисляет количество материала, необходимое для производства одной единицы продукции, и затем умножает это значение на количество продукции с учетом процента брака.

1. Интеграция DLL в приложение

После завершения разработки DLL, она была интегрирована в основное приложение. Для этого:

* Библиотека была скомпилирована в DLL.
* Ссылка на созданную DLL была добавлена в проект приложения.
* Класс CalculationClass и его метод GetQuantityForProduct использовались для выполнения расчетов.

## **Тестирование**

В данном разделе описываются разработанные юнит-тесты для класса CalculationClass, который отвечает за расчет необходимого количества материалов для производства продукции. Тесты проверяют корректность работы метода GetQuantityForProduct в различных сценариях, включая как нормальные, так и крайние случаи.

1. Структура юнит-тестов

Юнит-тесты организованы в классе UnitTests, который использует атрибуты из библиотеки Microsoft.VisualStudio.TestTools.UnitTesting. Каждый тест проверяет определенный аспект работы метода GetQuantityForProduct.

1. Тесты на некорректные входные данные

Тест на несуществующий тип продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NonExistentProductType()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(8, 1, 15, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на несуществующий тип материала:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NonExistentMaterialType()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 8, 15, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательное количество продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductQuantity()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, -4, 20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательную ширину продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductWidth()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, 15, -20, 45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на отрицательную длину продукции:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NegativeProductLength()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 4, 15, 20, -45);  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

1. Тесты на корректные входные данные

Тесты на корректные расчеты для различных типов продукции и материалов:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_MaterialType1ProductType1\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(1, 1, 15, 20, 45);  int correctResult = 34897;  Assert.AreEqual(actualResult, correctResult);  }  [TestMethod]  public void GetQuantityForProduct\_MaterialType2ProductType1\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(2, 1, 15, 20, 45);  int correctResult = 76477;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на случай, когда ширина и длина равны нулю:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_ZeroWidthAndLength\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 15, 0, 0);  int correctResult = 0;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на малые значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_SmallValues\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 1, 1, 1);  int correctResult = 7;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на большие значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_LargeProduct\_CorrectResult()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(3, 2, 2, 500, 500);  int correctResult = 4231500;  Assert.AreEqual(actualResult, correctResult);  } |

1. Тесты на тип возвращаемого значения

Тест на возврат значения -1 при передаче null:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_NullValue\_ReturnMinusOne()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(Convert.ToInt32(null), Convert.ToInt32(null),  Convert.ToInt32(null), Convert.ToInt32(null), Convert.ToInt32(null));  int correctResult = -1;  Assert.AreEqual(actualResult, correctResult);  } |

Тест на тип возвращаемого значения:

|  |
| --- |
| [TestMethod]  public void GetQuantityForProduct\_ReturnInt()  {  CalculationClass calculation = new CalculationClass();  int actualResult = calculation.GetQuantityForProduct(2, 1, 15, 20, 45);  Assert.IsInstanceOfType<int>(actualResult);  } |

Также был протестирован экран редактирования партнёра.

Список тестов:

1. Редактирование партнера с указанием отрицательного рейтинга;
2. Редактирование партнера с полной информацией;
3. Редактирование партнера с указанием рейтинга нечисловыми значениями;
4. Редактирование партнера без информации;
5. Кнопка редактирования изменяет цвет при нажатии;

Подробнее с ui-тестами можно ознакомиться в Приложении \*

## **Разработка UML диаграмм.**

Для системы фитнес-центра были разработаны две диаграммы.

Диаграмма использования иллюстрирует взаимодействие пользователей (клиентов, тренеров и администраторов) с системой, как показано на рисунке №\*. Неавторизованный пользователь имеет возможность только авторизоваться в системе. Клиент, в свою очередь, может просмотреть список тренеров и оставить заявку, которая обязательно должна содержать указание цели. После этого клиент может просматривать свои заявки, доступные занятия и отслеживать свой прогресс. Администратор имеет возможность добавлять новых тренеров в систему, записывать их на курсы и предоставлять скидки постоянным клиентам. Тренер может просматривать список своих текущих клиентов, а также новые заявки, которые он может принять или отклонить с указанием причины.

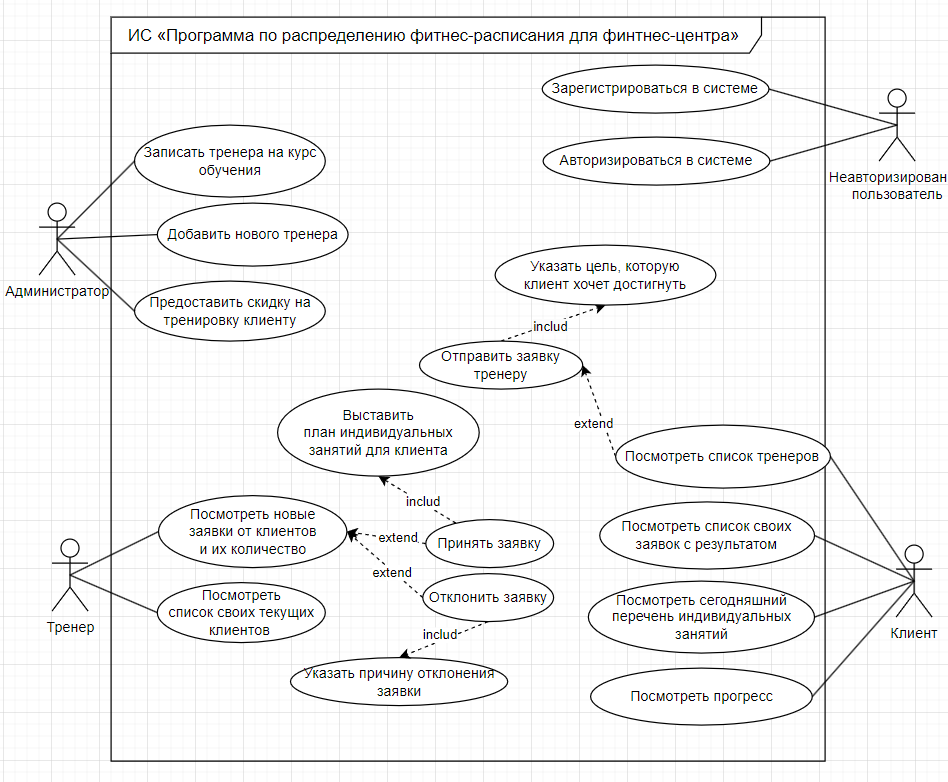


Рисунок №\* – Диаграмма использования.

Диаграмма деятельности иллюстрирует процессы, такие как выбор тренера и получение плана тренировок, и представлена на рисунке №\*. После авторизации в системе клиент может выбрать тренера и отправить ему заявку. В ответ на это действие система одновременно отображает список заявок для клиента и отправляет заявку тренеру, который может ее просмотреть. Тренер принимает решение либо отклонить заявку, указав причину отказа, после чего система обновляет список заявок у клиента, либо принять заявку и выставить план тренировок. В этом случае система отображает пользователю две новые вкладки, позволяющие ознакомиться с перечнем индивидуальных занятий или планом тренировок.

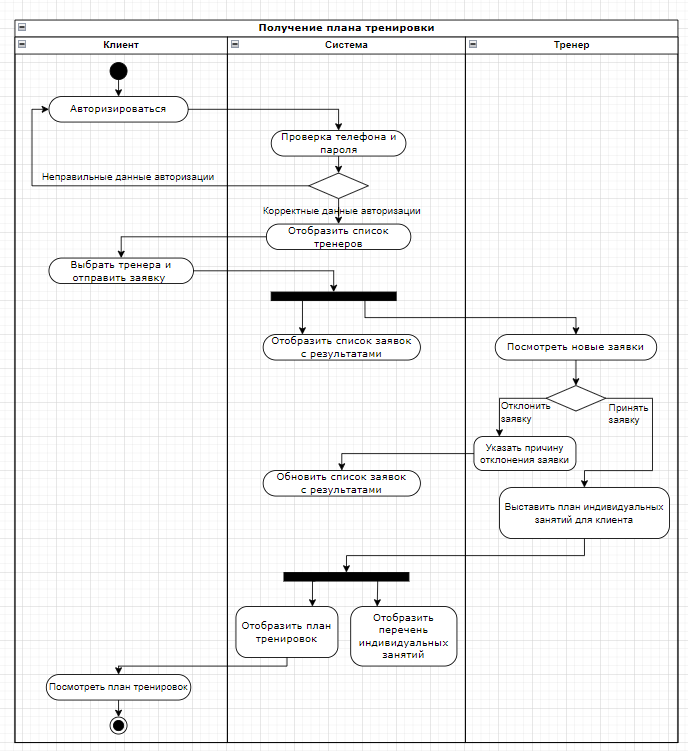


Рисунок №\* – Диаграмма деятельности.

# **Разработка информационной системы «Пиши-Стирай»**

## **Разработка базы данных, объектов базы данных и импорт**

При помощи имеющегося скрипта созданы основные таблицы, после просмотра данных для импорта база данных была доработана (добавлены сущности, атрибуты и соответствующие связи). Таким образом база данных была приведена в третью нормальную форму. На диаграмме 1 представлена ER-диаграмма, которая отражает сущности и связи разработанной базы данных для ООО «Пиши-Стирай».

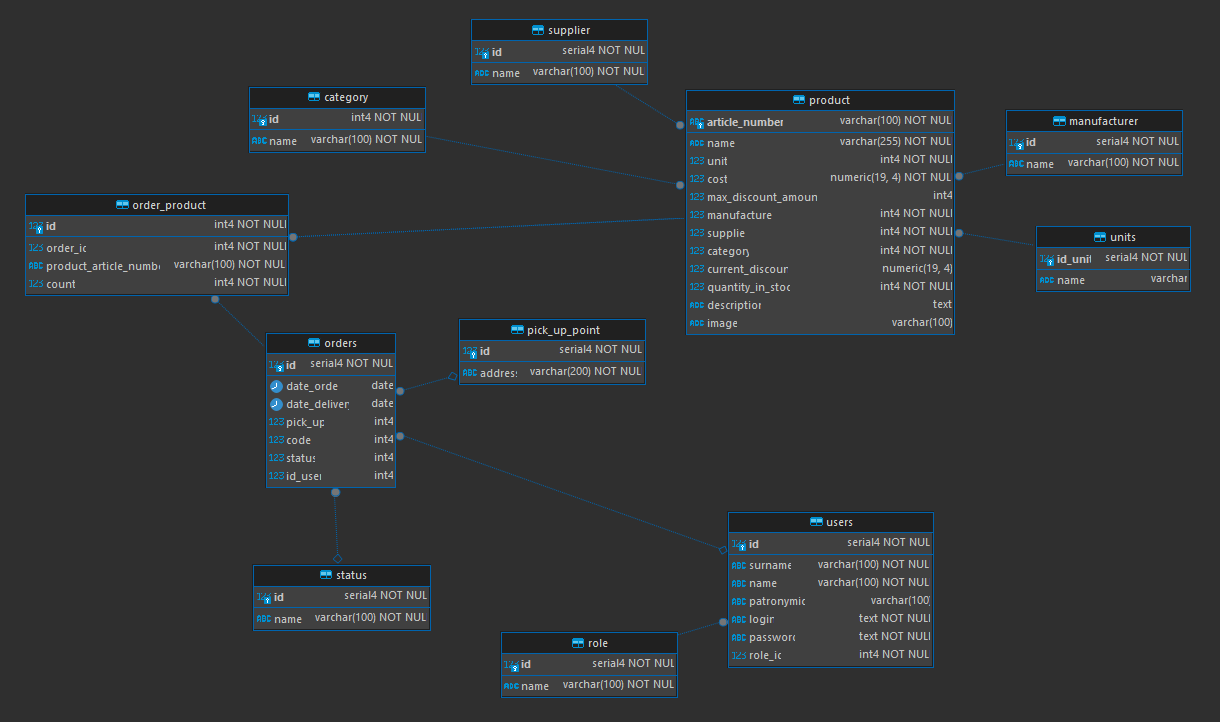


Рисунок №1 – ER-диаграмма базы данных

Скрипт для создания данной базы данных находиться в приложении Б Табл Табл. 1. Данные для импорта приведены к необходимому формату в программе Excel. На рисунках 2-12 представлены заполненные таблицы данными для импорта.

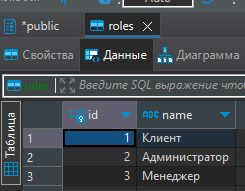


Рисунок №2 – Таблица ролей пользователей

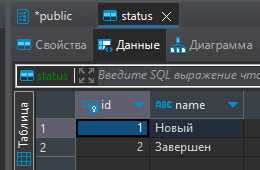


Рисунок №3 – Таблица статусы заказа

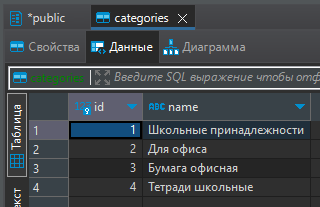


Рисунок №4 – Таблица категорий товаров

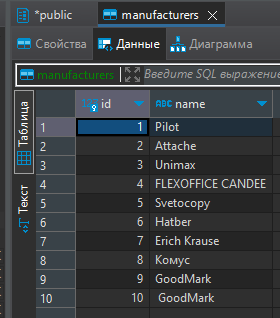


Рисунок №5 – Таблица производителей

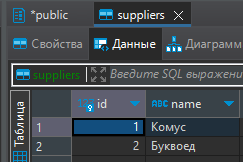


Рисунок №6 – Таблица поставщиков

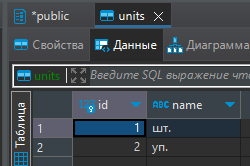


Рисунок №7 – Таблица единиц измерения

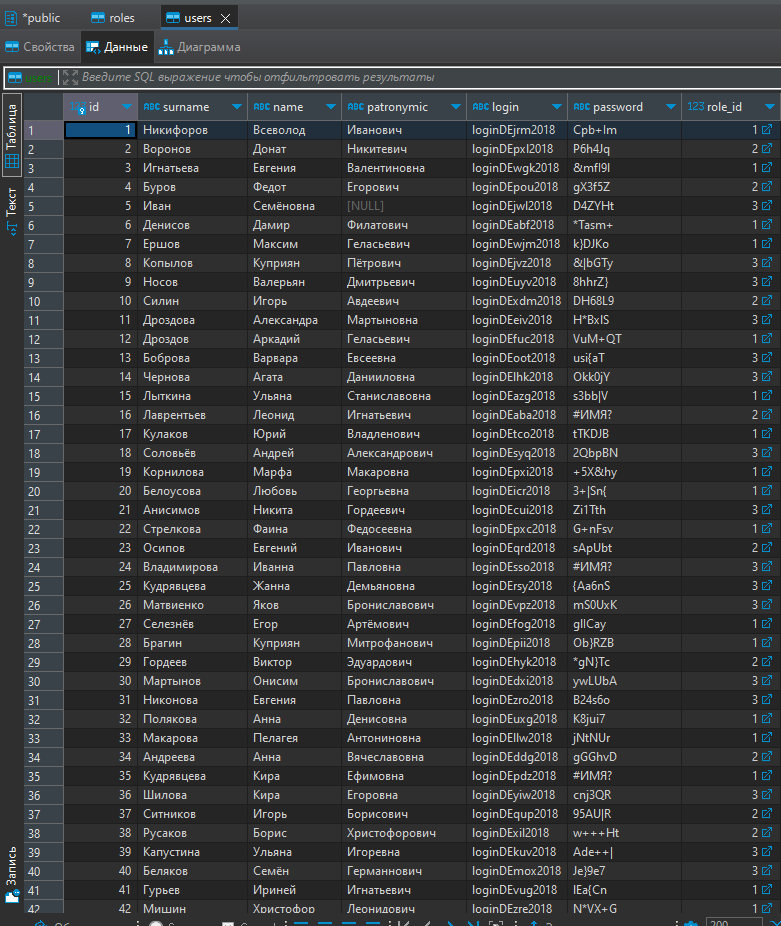


Рисунок №8 – Таблица пользователей

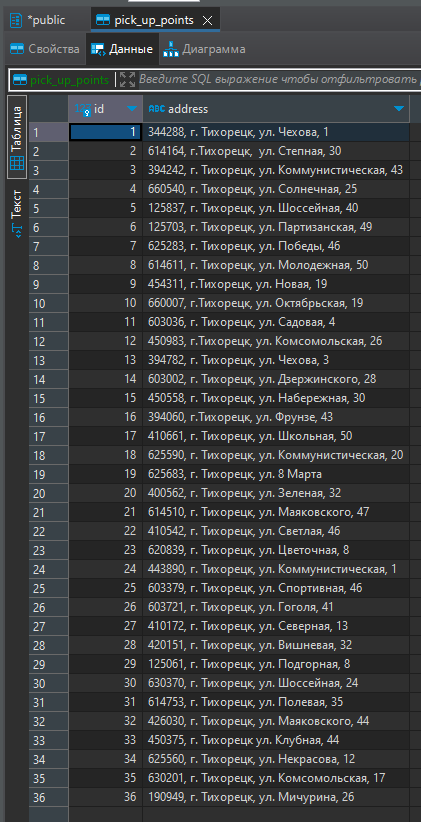


Рисунок №9 – Таблица пунктов выдачи

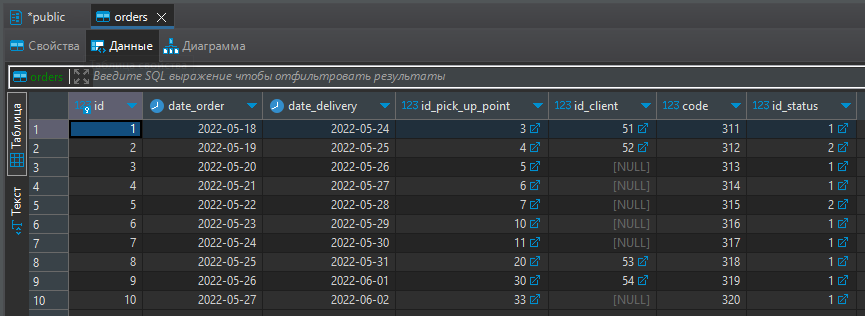


Рисунок №10 – Таблица заказов

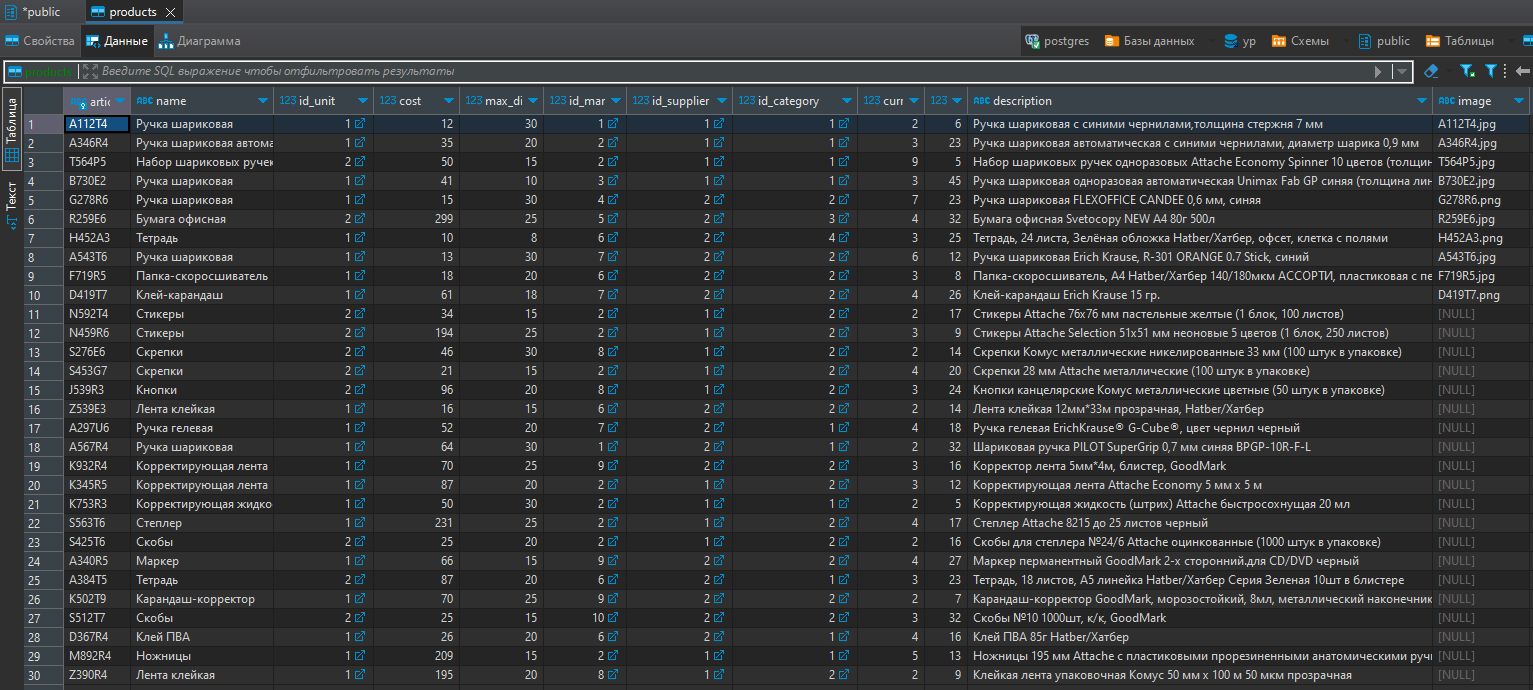


Рисунок №11 – Таблица товаров

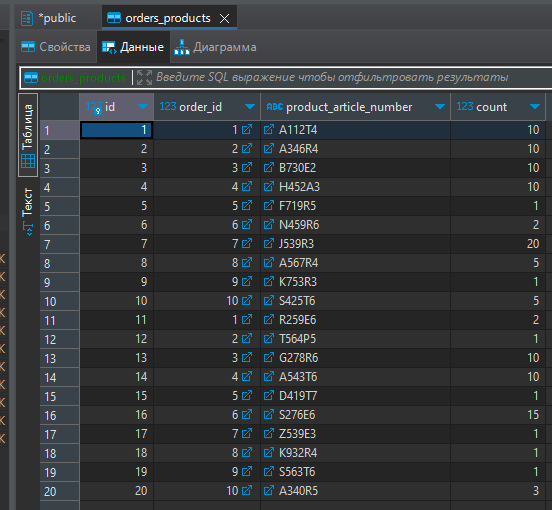


Рисунок №12 – Таблица товаров в заказе

## **Разработка настольного приложения**

1. Авторизация

В процессе разработки создана страница входа, которая появляется при запуске приложения. На этой странице пользователю предлагается ввести логин и пароль, а также возможность войти в систему в роли гостя.

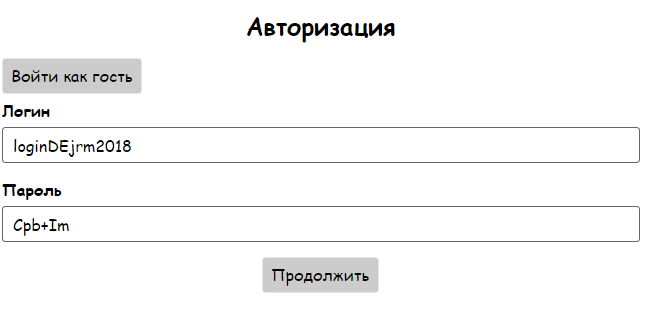


Рисунок №13 – Авторизация

После первой неуспешной попытки авторизации выводится сообщение о неудаче, а также появляется поле для ввода CAPTCHA (состоящей из 4 символов, включающих цифры и буквы латинского алфавита, которые могут быть перечеркнуты).

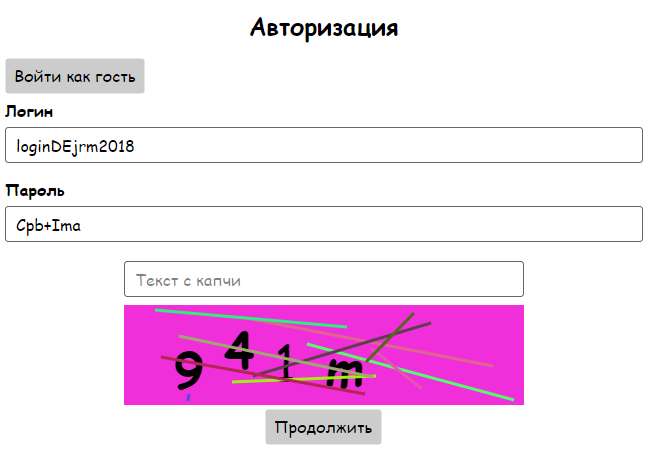


Рисунок №14 – CAPTCHA

После попытки неудачной авторизации с вводом CAPTCHA система блокирует возможность дальнейшего входа на 10 секунд.

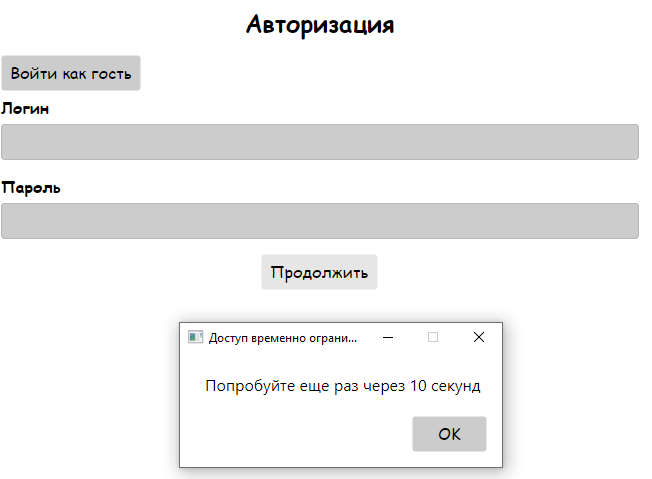


Рисунок №15 – Блокировка

1. Список товаров

Реализован поиск, сортировка и фильтрация списка товаров, происходящая совместно и в реальном времени без нажатия дополнительных кнопок. При нажатии на элемент правой кнопкой мыши появляется контекстное меню с возможностью добавить продукт к заказу в количестве 1 шт. При повторном нажатии количество продукта в заказе увеличится. Если в корзине есть хотя бы один продукт – появляется кнопка «Посмотреть заказ».

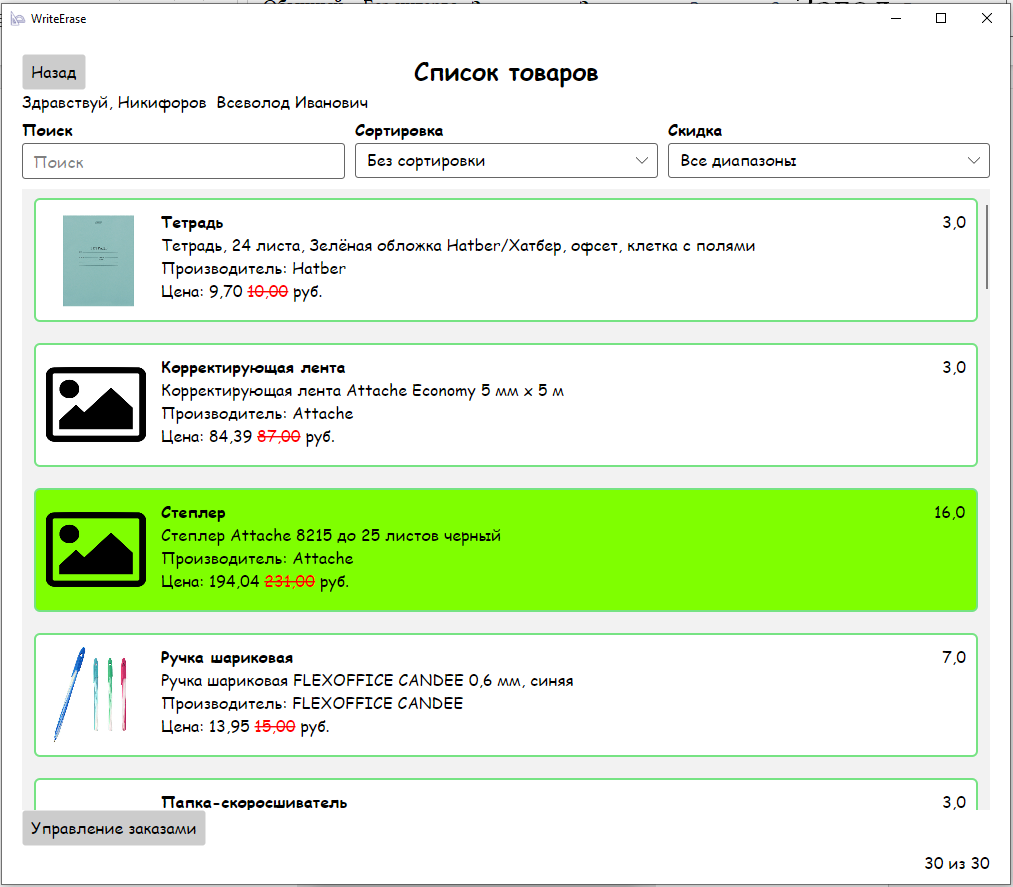


Рисунок №16 – Список продуктов

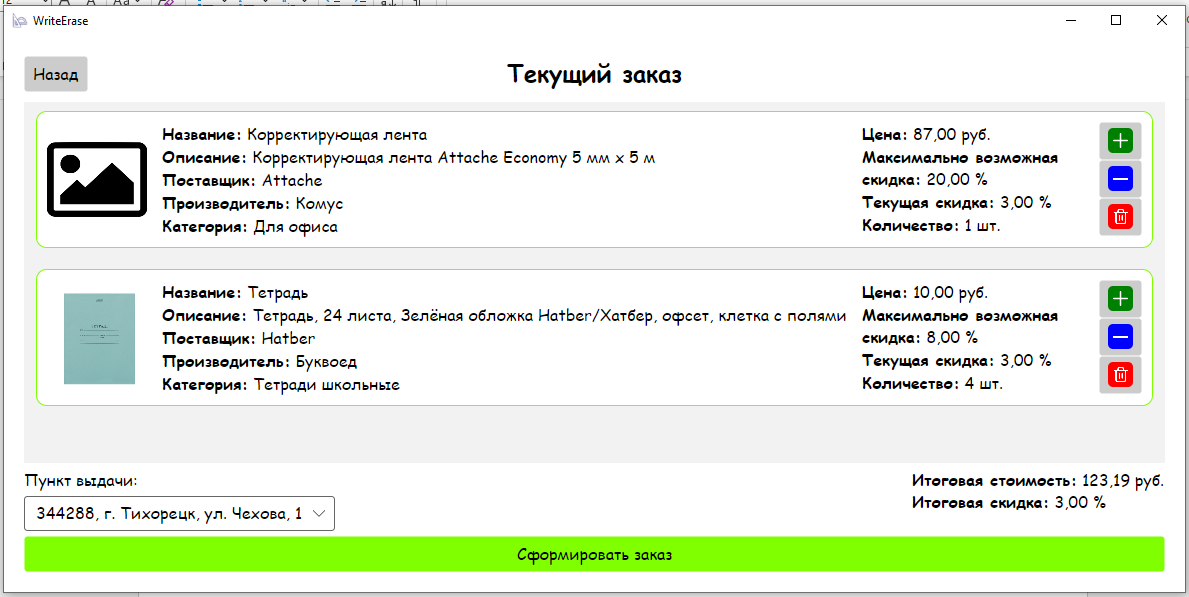


Рисунок №17 – Заказ

Если у товара снижена стоимость, то основная цена перечеркнута, и рядом с ней указана итоговая стоимость. В нижней части окна отображается количество выведенных данных и общее количество записей в базе. В случае, если данные в таблицу выводятся после фильтрации или поиска, количество выведенных данных обновляется исходя из размера выборки.

Для администратора и менеджера в системе реализован функционал работы с заказами. На странице «Управление заказами» присутствует сортировка заказа по убыванию и возрастанию стоимости, а также фильтрация по размеру суммарной скидки. Если все товары в заказе есть на складе в наличии более 3 позиций, то строка с заказом выделена цветом #20b2aa. Если хотя бы одного товара нет на складе, то строка выделена цветом #ff8c00.

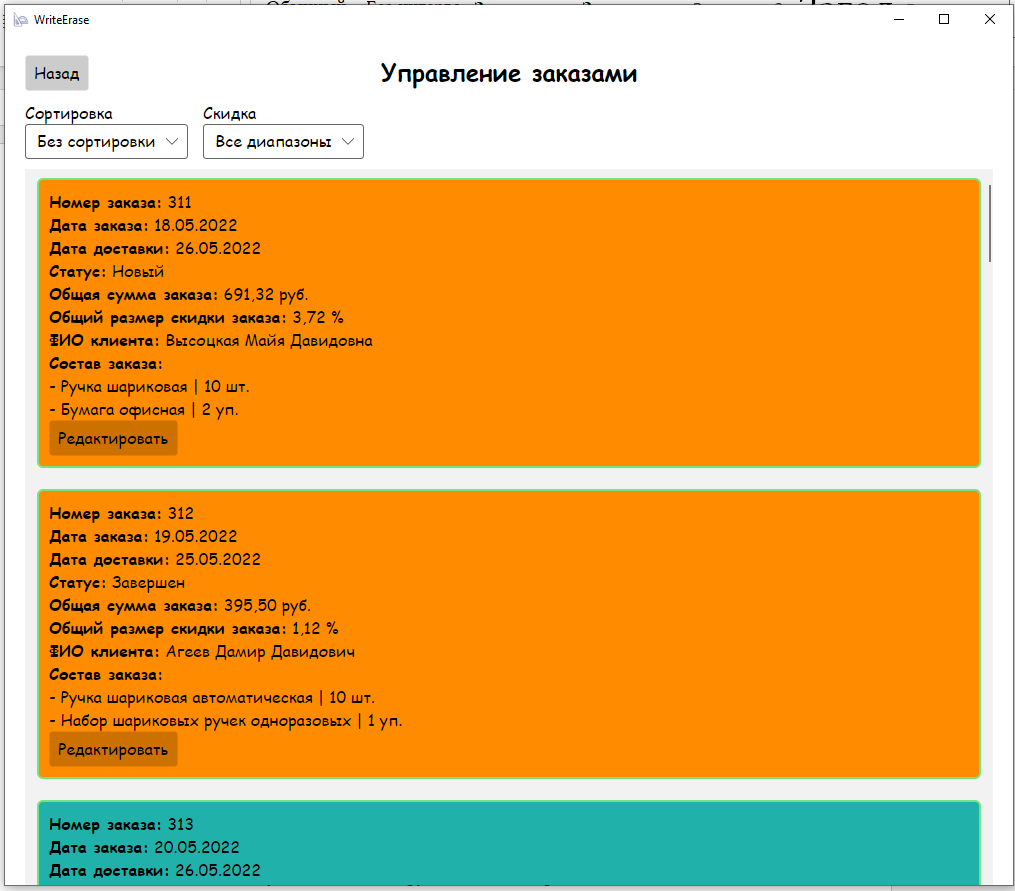


Рисунок №18 – Заказы клиентов

При нажатии на заказ открывается окно изменения заказа, в котором возможно поменять дату доставки и статус заказа (выпадающий список со статусами заказа из базы данных). При изменении информации о заказе отображается сообщение об изменении данных заказа и осуществляется переход на страницу всех заказов.

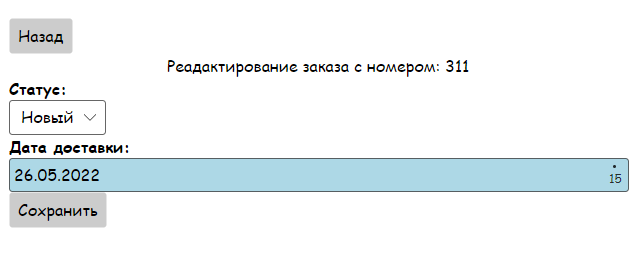


Рисунок №19 – Изменение заказа

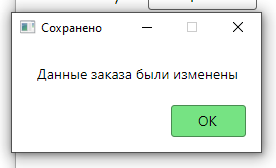
****

Рисунок №20 – Сообщение о сохранении изменений

## **Разработка библиотеки**

Библиотека имеет один метод «AvailablePeriods», который принимает такие данные, как массивы начальных времен (startTimes) и их продолжительности (durations), начало и конец рабочего дня (beginWorkingTime, endWorkingTime), продолжительность консультации (consultationTime). На выходе возвращается массив строк, представляющий доступные временные интервалы.

Код библиотеки находится в Прил. В, Табл. 1.

В начале метода идет проверка на корректность переданных значений. Далее определяется список занятых интервалов. После этого интервалы сортируются по времени начала для создания списка свободных интервалов. После этого проверяются интервалы от начала рабочего дня до первого занятого интервала, таким образом в лист свободных интервалов добавляются новые интервалы. Полученный список преобразуется в массив строк.

## **Тестирование библиотеки**

Для разработки библиотеки разрабатывались 10 Unit-тестов, которые охватывают все основные исключения и написаны по технике TDD. Эти тесты можно увидеть в Прил. Г Табл. 1. Вот краткое описание каждого теста:

1. «AvailablePeriods\_NoBusyIntervals\_ReturnsFullDay» проверяет, что если нет занятых интервалов, то возвращаются все возможные интервалы в течение рабочего дня.
2. «AvailablePeriods\_OneBusyInterval\_ReturnsCorrectFreeIntervals» проверяет корректность возвращаемых свободных интервалов при наличии одного занятого интервала в середине дня.
3. «AvailablePeriods\_BusyIntervalAtStart\_ReturnsCorrectFreeIntervals» проверяет корректность возвращаемых свободных интервалов при наличии занятого интервала в начале дня.
4. «AvailablePeriods\_BusyIntervalAtEnd\_ReturnsCorrectFreeIntervals» проверяет корректность возвращаемых свободных интервалов при наличии занятого интервала в конце дня.
5. «AvailablePeriods\_OverlappingBusyIntervals\_ReturnsCorrectFreeIntervals» проверяет корректность возвращаемых свободных интервалов при наличии нескольких пересекающихся занятых интервалов.
6. «AvailablePeriods\_ConsultationTimeTooLong\_ReturnsCorrectInterval» проверяет, если минимальное время консультации больше, чем доступное свободное время, возвращаются корректные интервалы.
7. «AvailablePeriods\_BeginWorkingTimeAfterEnd\_ReturnsError» проверяет, что при некорректных входных данных (начало рабочего дня позже его окончания) возвращается ошибка.
8. «AvailablePeriods\_InvalidConsultationTime\_ReturnsError» проверяет, что при некорректных входных данных (минимальное время <= 0) возвращается ошибка.
9. «AvailablePeriods\_ArraysLengthMismatch\_ReturnsError» проверяет, что при несовпадении длин массивов startTimes и durations возвращается ошибка.
10. «AvailablePeriods\_FullDayBusy\_ReturnsEmptyArray» проверяет, если занятые интервалы покрывают весь день, возвращается пустой массив.

## **Тестовая документация**

Реализована тестовая документация для тестирования функции добавления товара администратором.

Аннотация тестов

|  |  |
| --- | --- |
| **Название проекта** | «WriteErase» |
| **Рабочая версия** | 1.0 |
| **Имя тестирующего** | Климычева Анастасия Артемовна |
| **Дата(ы) теста** | 14.03.2025 |

Тестовый пример #1:

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_AP\_1: Full Field Product Creation |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка добавления товара администратором с заполнением всех обязательных полей |
| **Краткое изложение теста** | Тест проверяет возможность добавления нового товара в систему при корректном заполнении всех полей формы. |
| **Этапы теста** | 1. Нажать на кнопку «Добавить товар». 2. Внести данные во все поля формы.   Нажать кнопку «Сохранить». |
| **Тестовые данные** | Данные о товаре |
| **Ожидаемый результат** | * Система отображает сообщение об успешном добавлении товара.   Товар сохраняется в базе данных и появляется в общем списке товаров. |
| **Фактический результат** | * Сообщение об успешном добавлении товара отображается.   Товар добавлен в базу данных и отображается в списке. |
| **Статус** | Зачет |
| **Предварительное условие** | Администратор авторизирован в системе |
| **Постусловие** | Возврат на страницу со списком товаров |
| **Примечания/комментарии** | - |

Тестовый пример #2:

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_AP\_2: Empty Field Product Addition |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка добавления товара администратором без заполнения полей |
| **Краткое изложение теста** | Тест проверяет реакцию системы при попытке добавить товар с пустыми полями формы. |
| **Этапы теста** | 1. Нажать кнопку «Добавить товар».  2. Нажать кнопку «Сохранить». |
| **Тестовые данные** | Отсутствуют |
| **Ожидаемый результат** | Пользователю отображается сообщение об ошибке: «Проверьте правильность заполнения полей и попробуйте еще раз». |
| **Фактический результат** | Пользователю отображается сообщение об ошибке: «Проверьте правильность заполнения полей и попробуйте еще раз». |
| **Статус** | Зачет |
| **Предварительное условие** | Администратор авторизирован в системе |
| **Постусловие** | - |
| **Примечания/комментарии** | - |

Тестовый пример #3:

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_AP\_3: Negative Price Input Test |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка добавления товара с отрицательной ценой |
| **Краткое изложение теста** | Тест проверяет, позволяет ли система ввести отрицательное значение в поле «Стоимость» при добавлении товара. |
| **Этапы теста** | 1. Нажать кнопку «Добавить товар».  2. Ввести в поле «Стоимость» отрицательное значение. |
| **Тестовые данные** | Данные о товаре (название, описание и т.д.), кроме поля «Стоимость», где указано отрицательное значение. |
| **Ожидаемый результат** | Поле «Стоимость» автоматически очищается и устанавливается значение 0, предотвращая ввод отрицательных чисел. |
| **Фактический результат** | Поле «Стоимость» очищается и отображает значение 0, блокируя ввод отрицательных значений. |
| **Статус** | Зачет |
| **Предварительное условие** | Администратор авторизирован в системе |
| **Постусловие** | - |
| **Примечания/комментарии** | - |

Тестовый пример #4:

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_AP\_4: Missing Product Name Test |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка добавления товара без указания названия |
| **Краткое изложение теста** | Тест проверяет, позволяет ли система добавить товар, если поле «Наименование» осталось пустым. |
| **Этапы теста** | 1. Нажать кнопку «Добавить товар».  2. Ввести данные о товаре, оставив поле «Наименование» пустым.  3. Нажать кнопку «Сохранить». |
| **Тестовые данные** | Данные о товаре (кроме поля «Наименование»). |
| **Ожидаемый результат** | Пользователь видит сообщение об ошибке: «Проверьте правильность заполнения полей и попробуйте еще раз». |
| **Фактический результат** | Пользователь видит сообщение об ошибке: «Проверьте правильность заполнения полей и попробуйте еще раз». |
| **Статус** | Зачет |
| **Предварительное условие** | Администратор авторизирован в системе |
| **Постусловие** | - |
| **Примечания/комментарии** | Поля «Артикул» и «Наименование» обязательны для добавления товара. |

Тестовый пример #5:

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_AP\_5: Non-Numeric Input Validation |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Проверка ввода нечисловых значений в числовое поле |
| **Краткое изложение теста** | Тест проверяет, позволяет ли система вводить символы и буквы в поле, предназначенное для числовых значений. |
| **Этапы теста** | 1. Нажать кнопку «Добавить товар».  2. Ввести в поле «Текущая скидка» (или другое поле, принимающее численное значение) символы и буквы |
| **Тестовые данные** | Символы и буквы (например, «abc», «!@#»). |
| **Ожидаемый результат** | Поле автоматически очищается, предотвращая ввод нечисловых значений. |
| **Фактический результат** | Поле очищается, блокируя ввод символов и букв. |
| **Статус** | Зачет |
| **Предварительное условие** | Администратор авторизирован в системе |
| **Постусловие** | - |
| **Примечания/комментарии** | - |

# **Разработка информационной системы Matule Me**

## **Разработка серверной части**

В рамках разработки серверной части мобильного приложения реализована интеграция с удалённой базой данных, что позволило обеспечить хранение, обработку и управление данными в облачной среде. Для этого был выбран сервис Supabase, который предоставляет мощные инструменты для работы с базами данных, аутентификации, хранилища файлов и других backend-функций.

В Supabase была создана база данных PostgreSQL, где определены таблицы, соответствующие бизнес-логике приложения по предоставленному в задании скрипту.

Схема БД представлена на Рисунке 1 ниже.

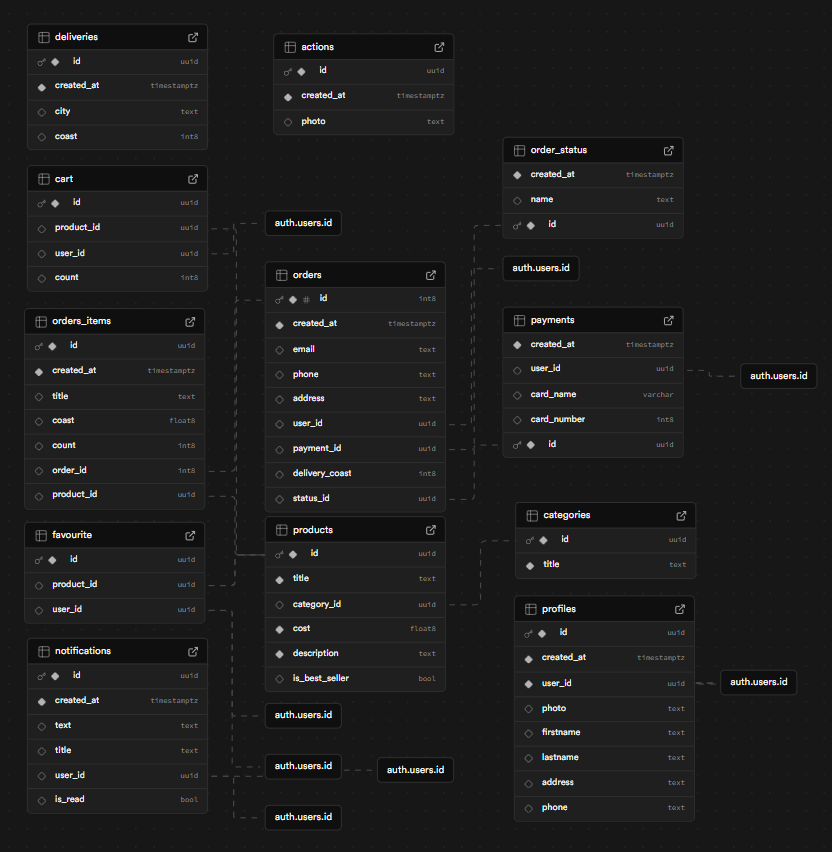


Рисунок № 1 – Схема БД.

## **Используемые библиотеки**

В процессе разработки мобильного приложения были использованы различные библиотеки, которые помогли реализовать ключевые функции и упростить процесс разработки. Ниже приведено описание каждой из них:

1. ZXing (com.google.zxing:core:3.4.1 и com.google.zxing:android-core:3.3.0) — для работы с QR-кодами и штрих-кодами. Они позволяют сканировать и генерировать коды, что было применено для авторизации пользователей и получения данных.
2. Dagger Hilt (com.google.dagger.hilt.android) — библиотека для внедрения зависимостей, которая упрощает управление зависимостями в приложении, такими как репозитории и сервисы. Это делает код более модульным и тестируемым.
3. Kotlin Serialization (org.jetbrains.kotlin.plugin.serialization) — плагин для сериализации и десериализации данных, например, JSON. Использовался для преобразования данных, полученных от сервера, в объекты Kotlin.
4. Jetpack Navigation (androidx.navigation:navigation-compose) — для реализации навигации между экранами в приложении на основе Jetpack Compose. Упрощает управление стеком навигации.
5. Supabase (io.github.jan-tennert.supabase) — open-source альтернатива Firebase. Использовался для работы с удалённой базой данных PostgreSQL, аутентификации пользователей и хранения файлов. Основные модули: postgrest-kt для RESTful API, gotrue-kt для аутентификации и ktor-client-android для сетевых запросов.
6. Kapt (kotlin-kapt) — плагин для обработки аннотаций в Kotlin. Использовался для работы с библиотеками, требующими генерации кода, например, Dagger Hilt.
7. Ktor (io.ktor:ktor-client-android) — асинхронный HTTP-клиент для Kotlin. Применялся для выполнения сетевых запросов в связке с Supabase.

Эти библиотеки помогли ускорить разработку, улучшить качество кода и реализовать ключевые функции приложения, такие как сканирование QR-кодов, навигация, управление зависимостями и взаимодействие с удалённой базой данных.

## **Архитектура приложения**

Приложение разработано с использованием Clean Architecture (чистой архитектуры), которая позволяет разделить код на слои с чёткими обязанностями и упрощает поддержку, тестирование и масштабирование проекта. Основная идея Clean Architecture — разделение приложения на независимые слои, где каждый слой имеет свою зону ответственности и зависит только от внутренних слоёв.

1. Слой Model

Слой Model отвечает за хранение и управление состояниями экранов. В этом слое были созданы модели состояний, которые описывают текущее состояние UI. Например, для экрана входа была создана модель, включающая поля для хранения email, пароля, ошибок и флагов, таких как открыто ли диалоговое окно. Эти модели использовались для передачи данных в UI-компоненты и обновления состояния экранов в зависимости от действий пользователя. Это позволило централизованно управлять данными и упростило реакцию на изменения в интерфейсе.

1. Слой Domain

Domain слой содержит бизнес-логику приложения и не зависит от внешних библиотек или платформ. В этом слое были реализованы объекты для работы с Supabase, включая создание клиента и выполнение запросов к базе данных. Также описаны сущности Supabase, которые отражают структуру данных, используемых в приложении. Дополнительно была настроена работа с локальным хранилищем для кэширования данных и обеспечения оффлайн-доступа. Это позволило изолировать бизнес-логику и сделать её легко тестируемой.

1. Слой Presentation

Presentation слой отвечает за отображение данных пользователю и обработку его действий. В этом слое были созданы UI-компоненты, такие как текстовые поля, кнопки и другие часто используемые элементы интерфейса. Также реализована навигация между экранами с использованием Jetpack Navigation. Для каждого экрана созданы ViewModel, которые управляют состоянием и логикой отображения. Дополнительно были написаны функции валидации данных, например, для проверки корректности введённого email или пароля. Это обеспечило удобное взаимодействие с пользователем и гибкость в управлении интерфейсом.

## **Модуль авторизации**

В модуль авторизации входят экраны: Авторизация, Регистрация, Забыл пароль, Верификация OTP, Новый пароль.

Для того, чтобы вёрстка экранов была проще, были вынесены в отдельные функции часто используемые элементы UI, такие как поля ввода пароля или ввода других значений, кнопки. Также была создана тема, в которой описаны стили для текста. Подробнее с функциями можно ознакомиться в Приложении Д.

В соответствии с [макетом](https://www.figma.com/design/t7yNd853ZuGihgKLNexZSr/UP-01.03?node-id=1-2&t=7U5UF6Zit4S5xnSy-1) создан экран Регистрации (см. Рис. \* ниже).

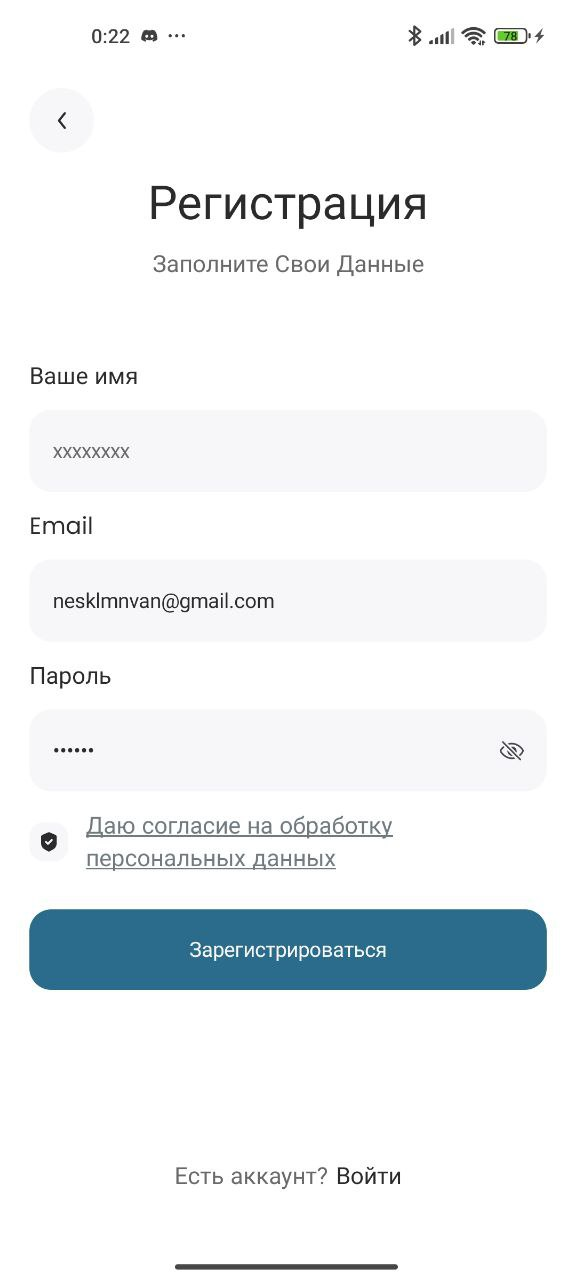


Рисунок № \* – Экран Регистрации.

На экране можно зарегистрироваться в системе. Также была реализована проверка email на корректность (соответствие паттерну «name@domenname.ru», где имя и доменное имя может состоять только из маленьких букв и цифр, старший домен только из символов количеством больше двух). При некорректном заполнении отображается ошибка в диалоговом окне (см. Рис. \* ниже).

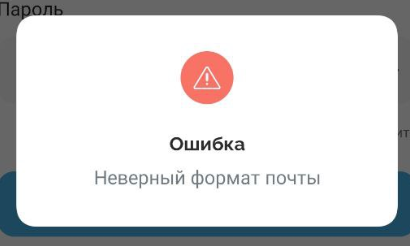


Рисунок № \* – Сообщение о некорректном формате почты.

Также у полей ввода пароля реализована возможность отображения пароля, и иконка при отображении и скрытии пароля выглядит в соответствии с макетом (см. Рис. \* ниже).

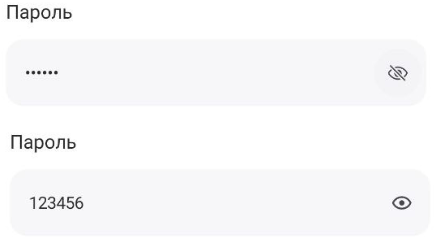


Рисунок № \* – Состояния отображения пароля.

Реализована валидация на пустоту полей. Выводится ошибка в диалоговом окне, если поля не заполнены (см. Рис. \* ниже).

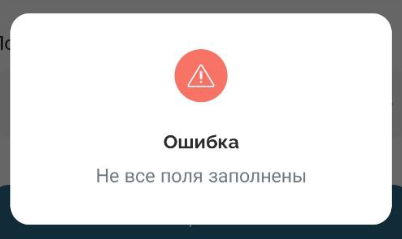


Рисунок № \* – Сообщение о недостаточном заполнении полей.

При нажатии на кнопку «Зарегистрироваться» реализована отправка запроса на сервер для регистрации с помощью почты и пароля. При успешной регистрации осуществляется переход на экран Авторизации. Перейти на экран Авторизации также можно, если нажать на текст-кнопку «Войти»

На экране авторизации (см. Рис. \* ниже) можно перейти на экран Регистрация, нажав на кнопку «Создать пользователя», а нажав на кнопку «Восстановить пароль», осуществляется переход на экран «Забыл пароль».

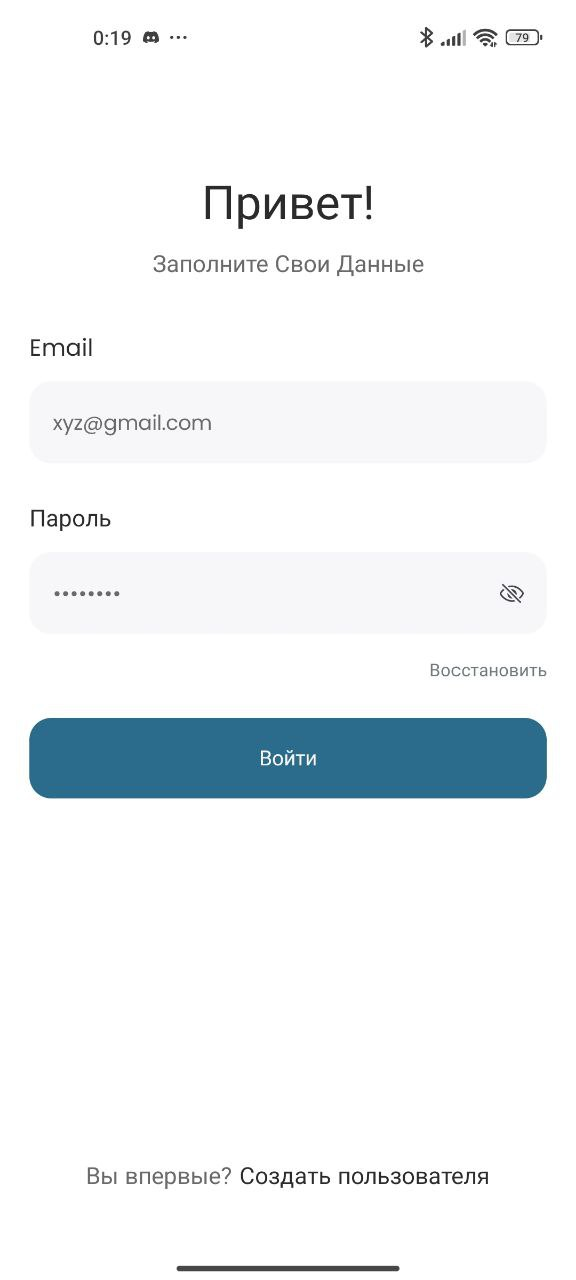


Рисунок № \* – Экран авторизации.

На экране «Забыл пароль» (см. Рис. \* ниже), при нажатии на кнопку «Отправить», при наличии в поле ввода корректного e-mail, отображается диалоговое окно, как на макете (см. Рис. \* ниже).

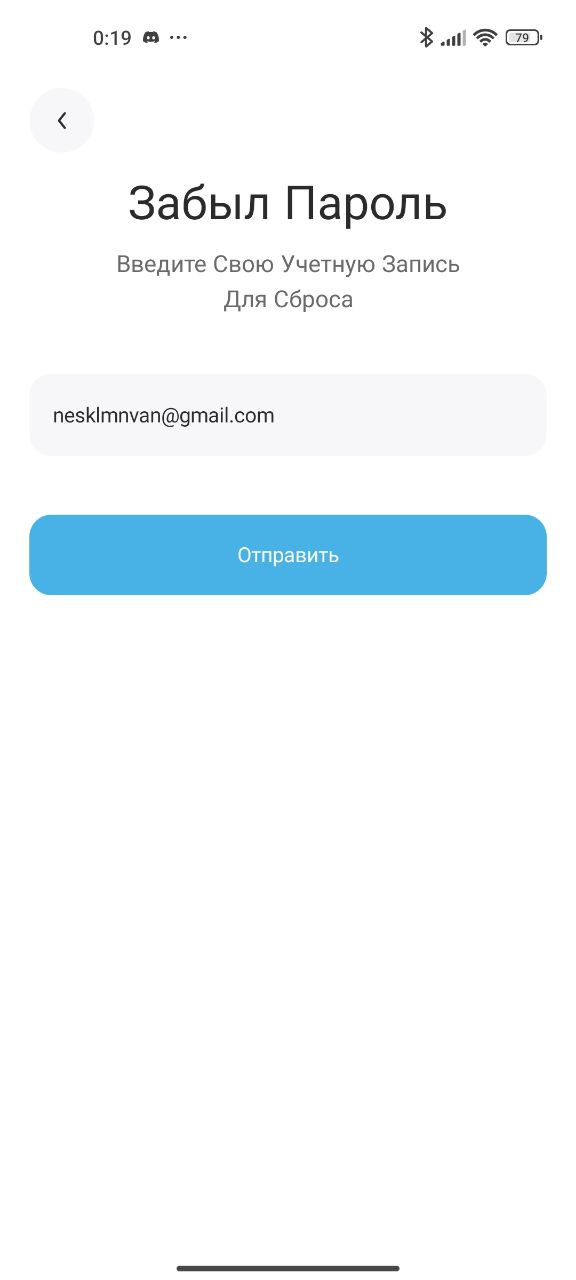


Рисунок № \* – Экран Забыл Пароль.

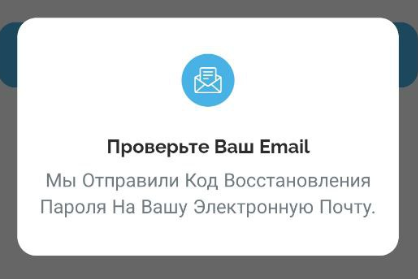


Рисунок № \* – Сообщение об отправке кода на почту.

При нажатии на диалоговое окно, осуществляется переход на экран «OTP Verification».

На экране «OTP Verification» (см. Рис. \* ниже) реализован механизм ввода 6 значного кода с переходом фокуса между текстовыми полями, а также таймер (60 секунд), по истечению которого можно отправить код ещё раз.

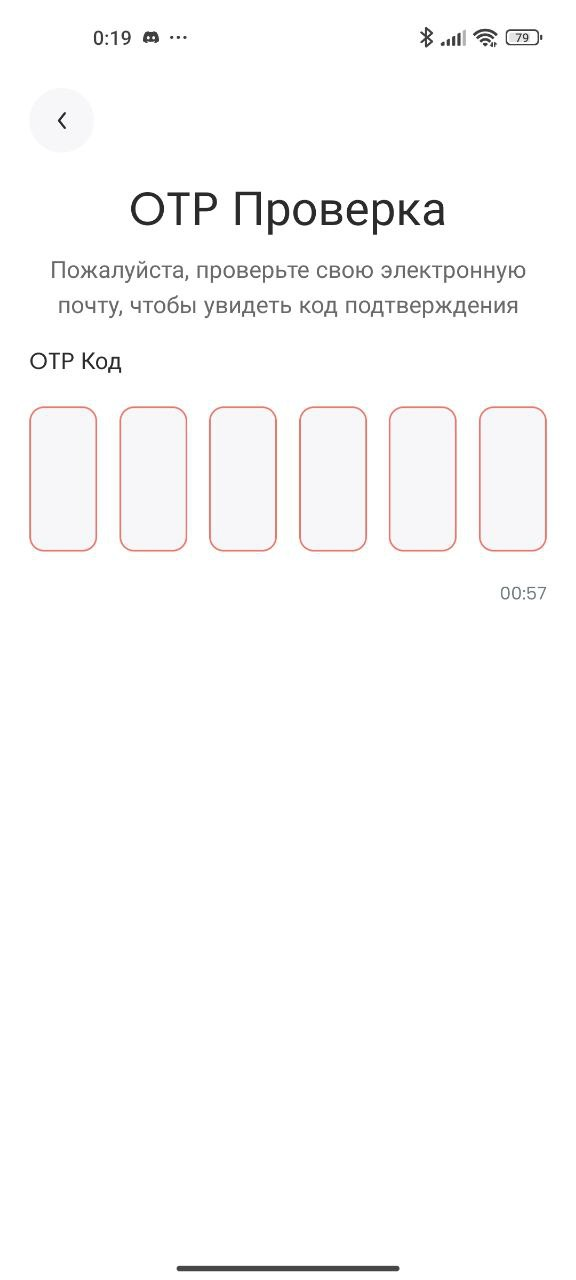


Рисунок № \* – Экран OTP Проверка.

Если код-пароль введен неверно, все квадраты становятся красными.

При верном вводе кода-пароля осуществляется переход на экран «Create New Password» (см. Рис. \* ниже), в котором реализована проверка на заполнение полей и на совпадение введённых паролей. После прохождения валидации полей осуществляется запрос на сервер на смену пароля и в случае отсутствия ошибок осуществляется переход на экран авторизации.

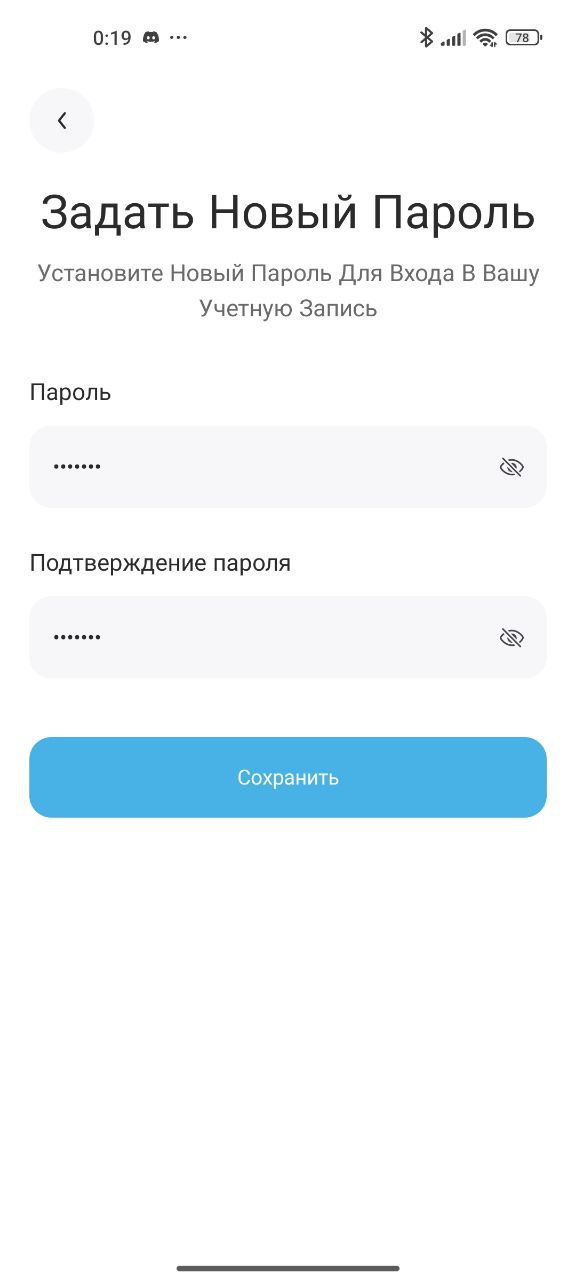


Рисунок № \* – Экран «Create New Password».

При успешной авторизации, осуществляется переход на экран «Home».

## **Основной модуль**

В основной модуль входят экраны, для доступа к которым необходимо авторизоваться в системе. Это экраны: Home, Favourite, Profile, SideMenu.

В соответствии с макетом был создан экран Home, на котором реализован вывод списка категорий, по нажатию на одну из которых, откроется отсортированный по выбранной категории список (см. Рис. \* ниже). Все данные на экране берутся с сервера, в том числе и акции.

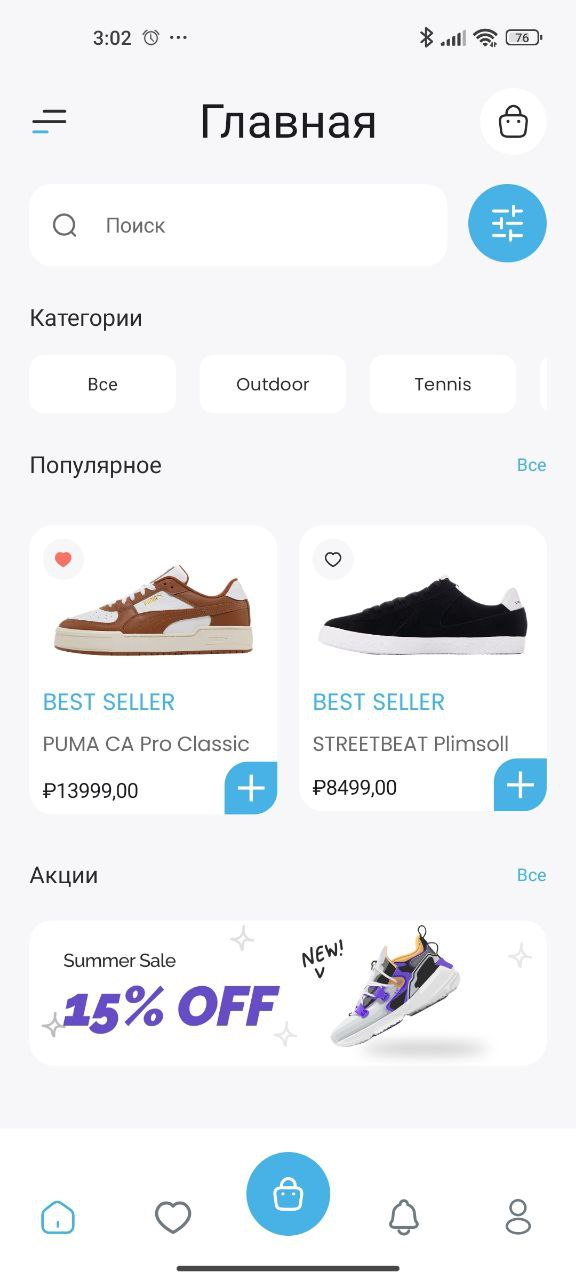


Рисунок № \* – Экран «Home».

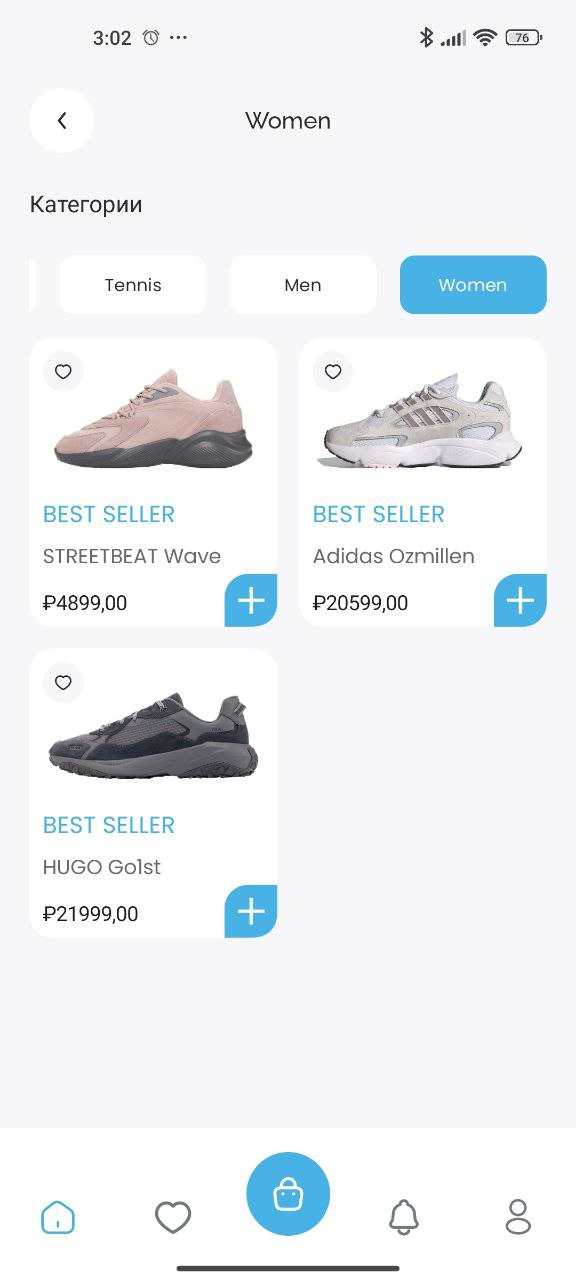


Рисунок № \* – Отфильтрованный список товаров.

Элемент карточки товара реализован как отдельный UI компонент, в который нужно передать объект класса продукта (см. Рис. \* ниже).

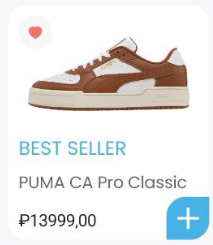


Рисунок № \* – Карточка товара.

Если в карточке товара нажать на иконку сердца, можно добавить или убрать товар из избранного. При этом, если иконка розового цвета, значит товар в избранном, если нет – товара в избранном нет (см. Рис. \* ниже).

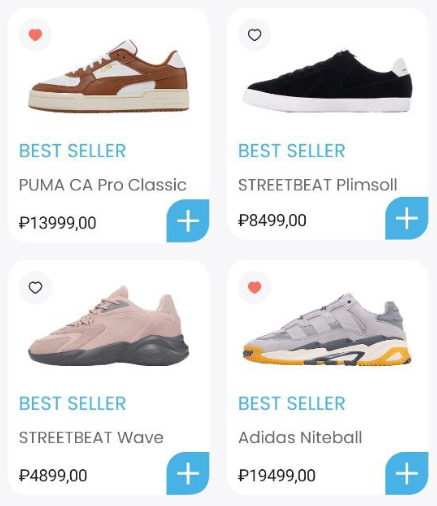


Рисунок № \* – Товары в избранном и нет.

Также на экране присутствует нижнее меню, в котором, по нажатию на определенный раздел, можно перейти в такие разделы, как: Favourite, Bucket, Notification, Profile (см. Рис. \* ниже).

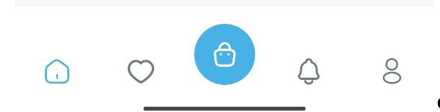


Рисунок № \* – Нижнее меню.

Если через нижнее меню перейти на экран Favourite, нажав на иконку сердца, отобразится список избранного (см. Рис. \* ниже).

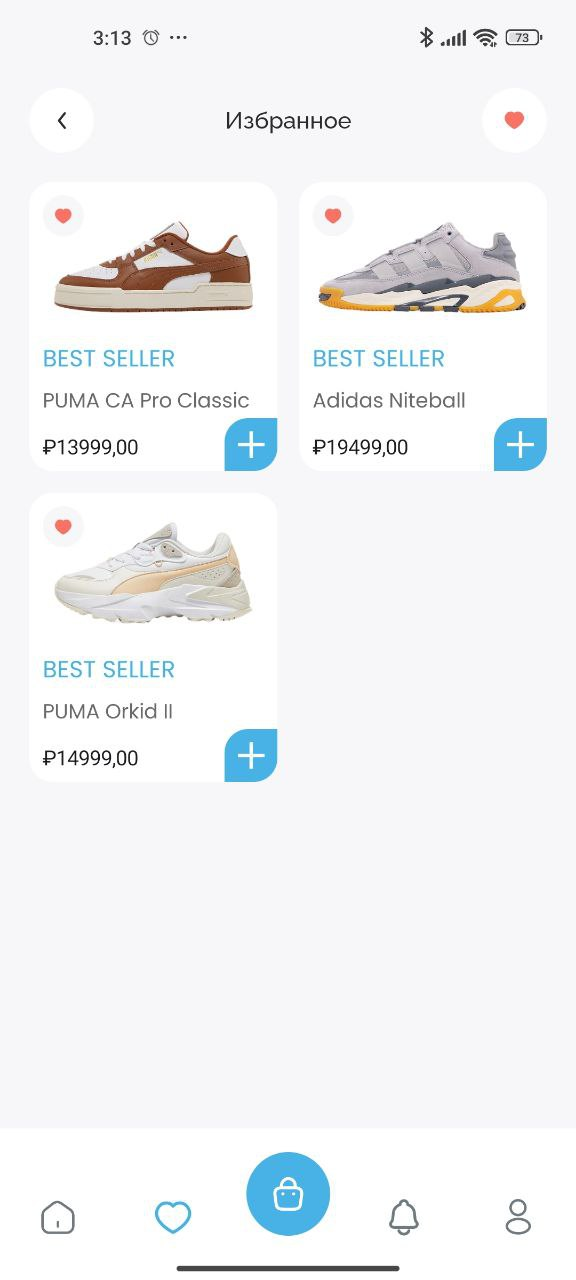


Рисунок № \* – Экран избранного.

Если через нижнее меню перейти на экран Profile, нажав на соответствующую иконку, отобразится экран профиля, на котором можно увидеть сгенерированный бар код id текущего пользователя и остальная информация о нём с возможностью. отредактировать (см. Рис. \* - \* ниже).

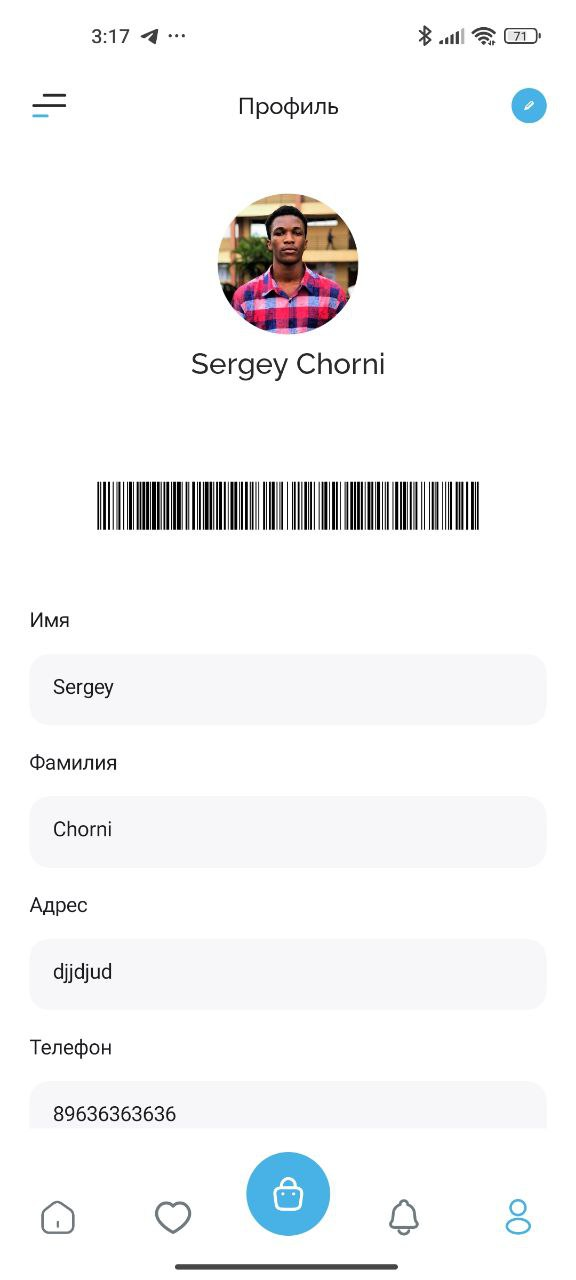


Рисунок № \* – Экран Profile, просмотр.

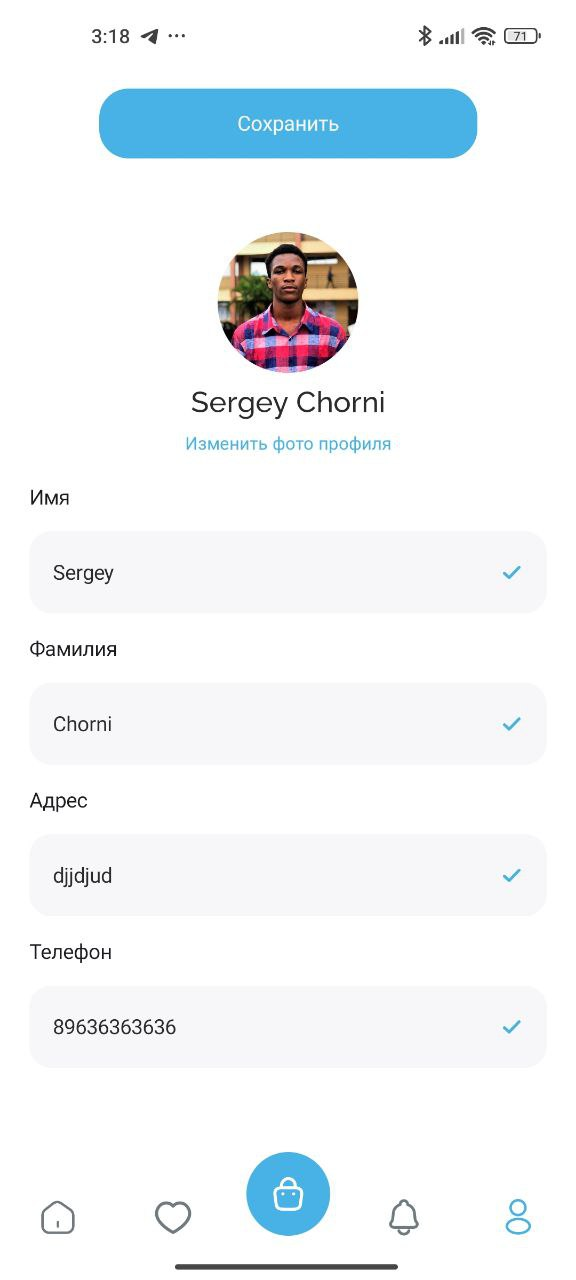


Рисунок № \* – Экран Profile, редактирование.

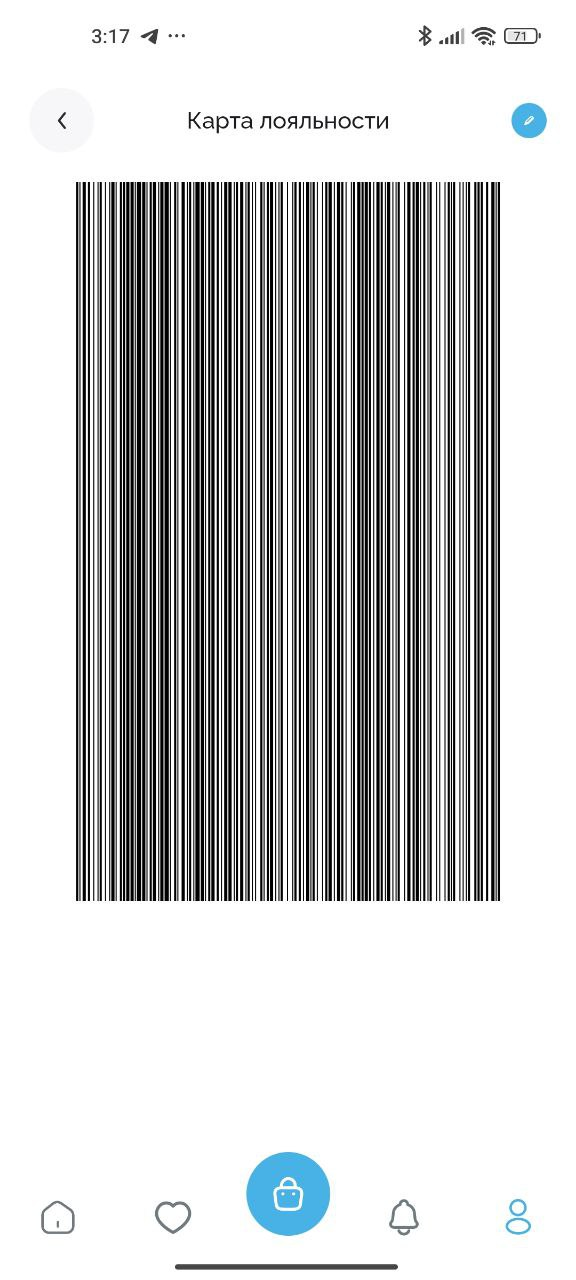


Рисунок № \* – Экран Profile, бар код.

Также, если на экране нажать на иконку меню в левом верхнем углу, откроется боковое меню, в котором пользователь может выйти из системы (см. Рис. \* ниже).

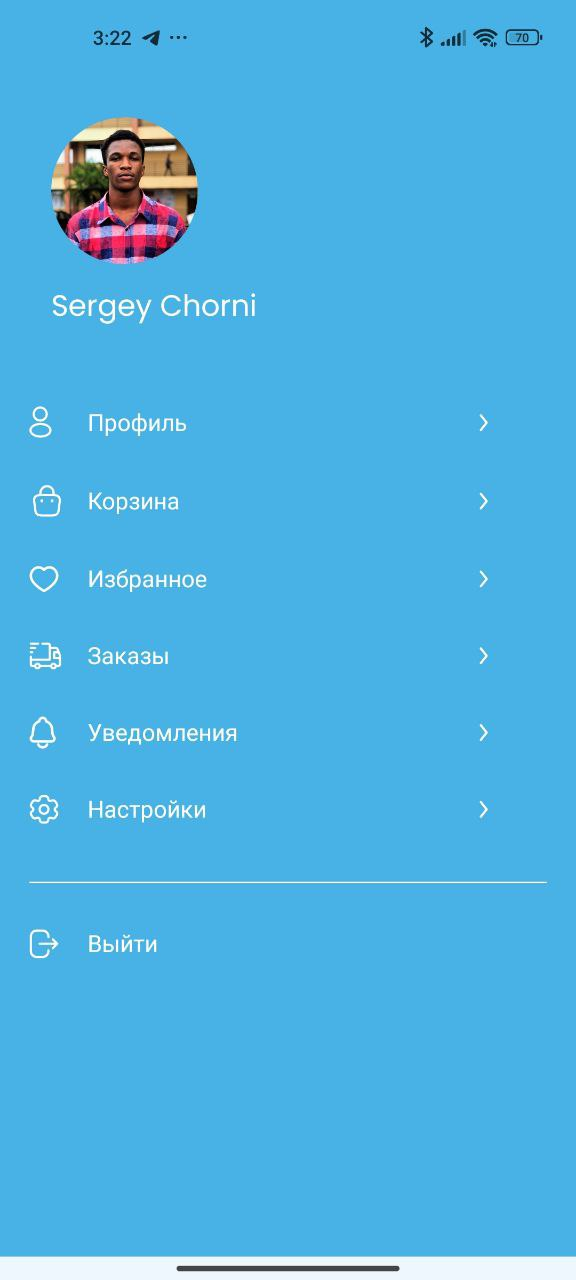


Рисунок № \* – Экран Side Menu.

# **Заключение**

В ходе учебной практики была успешно достигнута поставленная цель – овладение практическими навыками разработки, тестирования и отладки модулей программного обеспечения, а также освоение современных методов и инструментов программирования.

В процессе работы были изучены принципы модульного программирования, архитектура программных систем и основные концепции объектно-ориентированного программирования. На практике эти знания были применены при разработке программных модулей на языке C# с использованием кроссплатформенного XAML-фреймворка Avalonia для платформы .NET. Это позволило не только закрепить теоретические знания, но и получить ценный опыт работы с современными технологиями.

Дополнительно были освоены инструменты и среды разработки, что способствовало эффективному созданию и интеграции программных компонентов. Реализовано взаимодействие между модулями программного обеспечения, что позволило оценить важность гибкости и масштабируемости архитектурных решений.

Особое внимание было уделено тестированию программных модулей, включая модульное и интеграционное тестирование, что повысило надёжность и качество разрабатываемого программного обеспечения. Кроме того, в ходе работы активно использовалась система контроля версий Git, что способствовало эффективному управлению процессом разработки.

Таким образом, в результате прохождения учебной практики были приобретены ценные практические навыки, необходимые для дальнейшей профессиональной деятельности в области программирования, разработки программных систем и мобильных приложений.

# **Приложения**

## **Приложение А.**

Таблица №1 – Скрипт БД

|  |
| --- |
| -- public.partner\_types определение  CREATE TABLE public.partner\_types (  id\_type serial4 NOT NULL,  type\_name varchar NULL,  CONSTRAINT partner\_types\_pk PRIMARY KEY (id\_type)  );  -- public.product\_types определение  CREATE TABLE public.product\_types (  id\_type serial4 NOT NULL,  type\_name varchar NULL,  ratio numeric NULL,  CONSTRAINT product\_types\_pk PRIMARY KEY (id\_type)  );  -- public.material\_types определение  CREATE TABLE public.material\_types (  id\_type serial4 NOT NULL,  percent\_marriage numeric NULL,  type\_name varchar NULL,  CONSTRAINT material\_types\_pk PRIMARY KEY (id\_type)  );  -- public.partners определение  CREATE TABLE public.partners (  id\_partner serial4 NOT NULL,  id\_type\_company int4 NULL,  name\_company varchar NULL,  legal\_address varchar NULL,  tax\_identification\_number varchar NULL,  director\_full\_name varchar NULL,  contact\_phone varchar NULL,  contact\_email varchar NULL,  rating int4 NULL,  logo varchar NULL,  CONSTRAINT partners\_pk PRIMARY KEY (id\_partner),  CONSTRAINT partners\_partner\_types\_fk FOREIGN KEY (id\_type\_company) REFERENCES public.partner\_types(id\_type) ON DELETE RESTRICT ON UPDATE RESTRICT  );  -- public.products определение  CREATE TABLE public.products (  id\_product serial4 NOT NULL,  id\_product\_type int4 NULL,  product\_name varchar NULL,  article\_number varchar NULL,  minimum\_price numeric DEFAULT 0 NULL,  length numeric DEFAULT 0 NULL,  width numeric DEFAULT 0 NULL,  height numeric DEFAULT 0 NULL,  weight\_without\_pack numeric DEFAULT 0 NULL,  weight\_with\_pack numeric DEFAULT 0 NULL,  CONSTRAINT products\_pk PRIMARY KEY (id\_product),  CONSTRAINT products\_product\_types\_fk FOREIGN KEY (id\_product\_type) REFERENCES public.product\_types(id\_type) ON DELETE CASCADE ON UPDATE CASCADE  );  -- public.partner\_products определение  CREATE TABLE public.partner\_products (  id serial4 NOT NULL,  id\_partner int4 NULL,  id\_product int4 NULL,  count\_product int4 NULL,  date\_of\_sale date NULL,  CONSTRAINT partner\_products\_pk PRIMARY KEY (id),  CONSTRAINT partner\_products\_partners\_fk FOREIGN KEY (id\_partner) REFERENCES public.partners(id\_partner) ON DELETE CASCADE ON UPDATE CASCADE,  CONSTRAINT partner\_products\_products\_fk FOREIGN KEY (id\_product) REFERENCES public.products(id\_product) ON DELETE CASCADE ON UPDATE CASCADE  ); |

## **Приложение Б.**

Таблица №1 – Скрипт БД

|  |
| --- |
| -- public.category определение  CREATE TABLE public.category (  id int4 DEFAULT nextval('categoty\_id\_seq'::regclass) NOT NULL,  "name" varchar(100) NOT NULL,  CONSTRAINT categoty\_pkey PRIMARY KEY (id)  );  -- public.manufacturer определение  CREATE TABLE public.manufacturer (  id serial4 NOT NULL,  "name" varchar(100) NOT NULL,  CONSTRAINT manufacturer\_pkey PRIMARY KEY (id)  );  -- public.pick\_up\_point определение  CREATE TABLE public.pick\_up\_point (  id serial4 NOT NULL,  address varchar(200) NOT NULL,  CONSTRAINT pick\_up\_point\_pkey PRIMARY KEY (id)  );  -- public."role" определение  CREATE TABLE public."role" (  id serial4 NOT NULL,  "name" varchar(100) NOT NULL,  CONSTRAINT role\_pkey PRIMARY KEY (id)  );  -- public.status определение  CREATE TABLE public.status (  id serial4 NOT NULL,  "name" varchar(100) NOT NULL,  CONSTRAINT status\_pkey PRIMARY KEY (id)  );  -- public.supplier определение  CREATE TABLE public.supplier (  id serial4 NOT NULL,  "name" varchar(100) NOT NULL,  CONSTRAINT supplier\_pkey PRIMARY KEY (id)  );  -- public.units определение  CREATE TABLE public.units (  "name" varchar NULL,  id\_unit serial4 NOT NULL,  CONSTRAINT units\_pk PRIMARY KEY (id\_unit)  );  -- public.product определение  CREATE TABLE public.product (  article\_number varchar(100) NOT NULL,  "name" varchar(255) NOT NULL,  unit int4 NOT NULL,  "cost" numeric(19, 4) NOT NULL,  max\_discount\_amount int4 NULL,  manufacturer int4 NOT NULL,  supplier int4 NOT NULL,  category int4 NOT NULL,  current\_discount numeric(19, 4) NULL,  quantity\_in\_stock int4 NOT NULL,  description text NULL,  image varchar(100) NULL,  CONSTRAINT product\_pkey PRIMARY KEY (article\_number),  CONSTRAINT product\_category\_fkey FOREIGN KEY (category) REFERENCES public.category(id),  CONSTRAINT product\_manufacturer\_fkey FOREIGN KEY (manufacturer) REFERENCES public.manufacturer(id),  CONSTRAINT product\_supplier\_fkey FOREIGN KEY (supplier) REFERENCES public.supplier(id),  CONSTRAINT product\_units\_fk FOREIGN KEY (unit) REFERENCES public.units(id\_unit) ON DELETE CASCADE ON UPDATE CASCADE  );  -- public.users определение  CREATE TABLE public.users (  id serial4 NOT NULL,  surname varchar(100) NOT NULL,  "name" varchar(100) NOT NULL,  patronymic varchar(100) NULL,  login text NOT NULL,  "password" text NOT NULL,  role\_id int4 NOT NULL,  CONSTRAINT users\_pkey PRIMARY KEY (id),  CONSTRAINT users\_role\_id\_fkey FOREIGN KEY (role\_id) REFERENCES public."role"(id)  );  -- public.orders определение  CREATE TABLE public.orders (  id serial4 NOT NULL,  date\_order date NULL,  date\_delivery date NULL,  pick\_up int4 NULL,  code int4 NULL,  status int4 NULL,  id\_user int4 NULL,  CONSTRAINT orders\_pkey PRIMARY KEY (id),  CONSTRAINT orders\_pick\_up\_fkey FOREIGN KEY (pick\_up) REFERENCES public.pick\_up\_point(id) ON DELETE CASCADE ON UPDATE CASCADE,  CONSTRAINT orders\_status\_fkey FOREIGN KEY (status) REFERENCES public.status(id),  CONSTRAINT orders\_users\_fk FOREIGN KEY (id\_user) REFERENCES public.users(id) ON DELETE CASCADE ON UPDATE CASCADE  );  -- public.order\_product определение  CREATE TABLE public.order\_product (  id int4 DEFAULT nextval('orderproduct\_id\_seq'::regclass) NOT NULL,  order\_id int4 NOT NULL,  product\_article\_number varchar(100) NOT NULL,  count int4 NOT NULL,  CONSTRAINT orderproduct\_pkey PRIMARY KEY (id),  CONSTRAINT order\_product\_product\_fk FOREIGN KEY (product\_article\_number) REFERENCES public.product(article\_number) ON DELETE CASCADE ON UPDATE CASCADE,  CONSTRAINT orderproduct\_order\_id\_fkey FOREIGN KEY (order\_id) REFERENCES public.orders(id) ON DELETE CASCADE ON UPDATE CASCADE,  CONSTRAINT orderproduct\_product\_article\_number\_fkey FOREIGN KEY (product\_article\_number) REFERENCES public.product(article\_number)  ); |

## **Приложение В.**

Таблица №1 –Код класса Calculations

|  |
| --- |
| namespace SF2022User\_NN\_Lib  {  public class Calculations  {  public string[] AvailablePeriods(TimeSpan[] startTimes, int[] durations, TimeSpan beginWorkingTime, TimeSpan endWorkingTime, int consultationTime)  {  if (beginWorkingTime >= endWorkingTime)  return new[] { "Начало рабочего дня должно быть раньше его окончания." };  if (consultationTime <= 0)  return new[] { "Минимальное время должно быть больше нуля." };  if (startTimes == null || durations == null || startTimes.Length != durations.Length)  return new[] { "Ошибка: массивы startTimes и durations должны быть не null и одинаковой длины." };  var busyIntervals = CreateBusyIntervals(startTimes, durations);  busyIntervals.Sort((a, b) => a.Start.CompareTo(b.Start));  return GenerateFreeIntervals(busyIntervals, beginWorkingTime, endWorkingTime, consultationTime);  }  private List<(TimeSpan Start, TimeSpan End)> CreateBusyIntervals(TimeSpan[] startTimes, int[] durations)  {  var intervals = new List<(TimeSpan Start, TimeSpan End)>();  for (int i = 0; i < startTimes.Length; i++)  {  intervals.Add((startTimes[i], startTimes[i].Add(TimeSpan.FromMinutes(durations[i]))));  }  return intervals;  }  private string[] GenerateFreeIntervals(List<(TimeSpan Start, TimeSpan End)> busyIntervals, TimeSpan beginWorkingTime, TimeSpan endWorkingTime, int consultationTime)  {  var freeIntervals = new List<string>();  var currentTime = beginWorkingTime;  while (currentTime.Add(TimeSpan.FromMinutes(consultationTime)) <= endWorkingTime)  {  var intervalEnd = currentTime.Add(TimeSpan.FromMinutes(consultationTime));  if (!IsIntervalBusy(currentTime, intervalEnd, busyIntervals))  {  freeIntervals.Add($"{currentTime:hh\\:mm}-{intervalEnd:hh\\:mm}");  }  currentTime = currentTime.Add(TimeSpan.FromMinutes(consultationTime));  }  return freeIntervals.ToArray();  }  private bool IsIntervalBusy(TimeSpan start, TimeSpan end, List<(TimeSpan Start, TimeSpan End)> busyIntervals)  {  return busyIntervals.Any(busy => start < busy.End && end > busy.Start);  }  }  } |

## **Приложение Г.**

Таблица №1 – Тесты класса Calculations

|  |
| --- |
| using SF2022User\_NN\_Lib;  namespace TestProject  {  [TestClass]  public class UnitTestCalculationClass  {  private readonly Calculations \_calculations;  public UnitTestCalculationClass()  {  \_calculations = new Calculations();  }  // Тест 1: Нет занятых интервалов, весь день свободен  [TestMethod]  public void AvailablePeriods\_NoBusyIntervals\_ReturnsFullDay()  {  var startTimes = new TimeSpan[] { };  var durations = new int[] { };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "09:00-09:30", "09:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00" }, result);  }  // Тест 2: Один занятый интервал в середине дня  [TestMethod]  public void AvailablePeriods\_OneBusyInterval\_ReturnsCorrectFreeIntervals()  {  var startTimes = new[] { new TimeSpan(10, 0, 0) };  var durations = new[] { 60 };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "09:00-09:30", "09:30-10:00", "11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00" }, result);  }  // Тест 3: Занятый интервал в начале дня  [TestMethod]  public void AvailablePeriods\_BusyIntervalAtStart\_ReturnsCorrectFreeIntervals()  {  var startTimes = new[] { new TimeSpan(9, 0, 0) };  var durations = new[] { 60 };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00" }, result);  }  // Тест 4: Занятый интервал в конце дня  [TestMethod]  public void AvailablePeriods\_BusyIntervalAtEnd\_ReturnsCorrectFreeIntervals()  {  var startTimes = new[] { new TimeSpan(16, 0, 0) };  var durations = new[] { 60 };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "09:00-09:30", "09:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00" }, result);  }  // Тест 5: Несколько занятых интервалов с пересечениями  [TestMethod]  public void AvailablePeriods\_OverlappingBusyIntervals\_ReturnsCorrectFreeIntervals()  {  var startTimes = new[] { new TimeSpan(10, 0, 0), new TimeSpan(10, 30, 0) };  var durations = new[] { 60, 30 };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(  new[] {  "09:00-09:30",  "09:30-10:00",  "11:00-11:30",  "11:30-12:00",  "12:00-12:30",  "12:30-13:00",  "13:00-13:30",  "13:30-14:00",  "14:00-14:30",  "14:30-15:00",  "15:00-15:30",  "15:30-16:00",  "16:00-16:30",  "16:30-17:00"  }, result);  }  // Тест 6: Минимальное время больше, чем доступное свободное время  [TestMethod]  public void AvailablePeriods\_ConsultationTimeTooLong\_ReturnsCorrectIntervals()  {  var startTimes = new[] { new TimeSpan(10, 0, 0) };  var durations = new[] { 60 };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 120;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "11:00-13:00", "13:00-15:00", "15:00-17:00" }, result);  }  // Тест 7: Некорректные входные данные (начало рабочего дня позже конца)  [TestMethod]  public void AvailablePeriods\_BeginWorkingTimeAfterEnd\_ReturnsError()  {  var startTimes = new TimeSpan[] { };  var durations = new int[] { };  var beginWorkingTime = new TimeSpan(17, 0, 0);  var endWorkingTime = new TimeSpan(9, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "Начало рабочего дня должно быть раньше его окончания." }, result);  }  // Тест 8: Некорректные входные данные (минимальное время <= 0)  [TestMethod]  public void AvailablePeriods\_InvalidConsultationTime\_ReturnsError()  {  var startTimes = new TimeSpan[] { };  var durations = new int[] { };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 0;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "Минимальное время должно быть больше нуля." }, result);  }  // Тест 9: Некорректные входные данные (массивы разной длины)  [TestMethod]  public void AvailablePeriods\_ArraysLengthMismatch\_ReturnsError()  {  var startTimes = new[] { new TimeSpan(10, 0, 0) };  var durations = new int[] { };  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new[] { "Ошибка: массивы startTimes и durations должны быть не null и одинаковой длины." }, result);  }  // Тест 10: Занятые интервалы покрывают весь день  [TestMethod]  public void AvailablePeriods\_FullDayBusy\_ReturnsEmptyArray()  {  var startTimes = new[] { new TimeSpan(9, 0, 0) };  var durations = new[] { 480 }; // 8 часов  var beginWorkingTime = new TimeSpan(9, 0, 0);  var endWorkingTime = new TimeSpan(17, 0, 0);  var consultationTime = 30;  var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  CollectionAssert.AreEqual(new string[] { }, result);  }  }  } |

## **Приложение Д.**

Таблица №1 – функции часто используемых компонентов UI

|  |
| --- |
| @SuppressLint("RestrictedApi", "StateFlowValueCalledInComposition")  @Composable  fun ButtonBack(controller: NavHostController) {  val lastScreenIsNotSplash = controller.currentBackStack.value.size != 2  Box(  modifier = Modifier.alpha(if (lastScreenIsNotSplash) 1f else 0f)  .clip(CircleShape)  .background(background)  .size(44.dp),  contentAlignment = Alignment.Center  ) {  Icon(  imageVector = ImageVector.vectorResource(R.drawable.icon\_arrow),  contentDescription = "",  tint = text,  modifier = Modifier.clickable(  interactionSource = remember { MutableInteractionSource() },  indication = null  ) {  if (lastScreenIsNotSplash) {  controller.navigateUp()  }  }  )  }  }  @Composable  fun DialogError(title: String, desc: String, onDismissRequest: () -> Unit) {  Dialog(onDismissRequest = {  onDismissRequest()  }) {  Column(  modifier = Modifier.fillMaxWidth().clip(RoundedCornerShape(16.dp)).background(block).padding(vertical = 30.dp, horizontal = 10.dp),  verticalArrangement = Arrangement.Center,  horizontalAlignment = Alignment.CenterHorizontally  ) {  Box(modifier = Modifier.clip(CircleShape).background(red).padding(10.dp), contentAlignment = Alignment.Center) {  Icon(imageVector = ImageVector.vectorResource(R.drawable.icon\_error), contentDescription = "",  modifier = Modifier.size(24.dp), tint = block)  }  SpacerHeight(24.dp)  Text(title, style = MatuleMeTheme.typography.titleDialog)  SpacerHeight(8.dp)  Text(desc, style = MatuleMeTheme.typography.descDialog)  }  }  }  @Composable  fun SpacerHeight(height: Dp) {  Spacer(modifier = Modifier.height(height))  }  @Composable  fun SpacerWidth(width: Dp) {  Spacer(modifier = Modifier.width(width))  }  @Composable  fun AuthTextFieldBase(value: String, placeholder: String, testTag: String, onValueChange: (String) -> Unit) {  TextField(  value = value,  onValueChange = {  onValueChange(it)  },  modifier = Modifier.fillMaxWidth().testTag(testTag),  colors = TextFieldDefaults.colors(  unfocusedTextColor = text,  unfocusedContainerColor = background,  focusedContainerColor = background,  focusedTextColor = text,  focusedIndicatorColor = Color.Transparent,  unfocusedIndicatorColor = Color.Transparent,  disabledIndicatorColor = Color.Transparent,  disabledContainerColor = background  ),  shape = RoundedCornerShape(15.dp),  singleLine = true,  placeholder = {  Text(  placeholder,  style = MatuleMeTheme.typography.authHintField  )  }  )  }  @Composable  fun AuthTextFieldPass(value: String, placeholder: String, testTag: String, onValueChange: (String) -> Unit) {  var passwordVisible by remember { mutableStateOf(false) }  TextField(  value = value,  onValueChange = { onValueChange(it) },  modifier = Modifier.fillMaxWidth().testTag(testTag),  colors = TextFieldDefaults.colors(  unfocusedContainerColor = background,  unfocusedTextColor = text,  focusedContainerColor = background,  focusedTextColor = text,  focusedIndicatorColor = Color.Transparent,  unfocusedIndicatorColor = Color.Transparent,  disabledIndicatorColor = Color.Transparent  ),  singleLine = true,  shape = RoundedCornerShape(15.dp),  placeholder = {  Text(  placeholder,  style = MatuleMeTheme.typography.authHintField  )  },  visualTransformation = if (passwordVisible) VisualTransformation.None else PasswordVisualTransformation(),  keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),  trailingIcon = {  val image = if (passwordVisible)  painterResource(R.drawable.eye\_open)  else painterResource(R.drawable.eye\_close)  IconButton(onClick = { passwordVisible = !passwordVisible }) {  Icon(painter = image, "", modifier = Modifier.size(17.dp))  }  }  )  } |